НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: « Підсистема розпізнавання та автентифікації обличь в системі оплати »

Виконав: студент *IV* курсу, групи *ІС-63*

Портяний Іван Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.т.н., доц. Жданова Олена Григорівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *ст. вик. Проскура Світлана Леонідівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *д.т.н., доц. Клименко Ірина Анатоліївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2020 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Портяному Івану Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту « Підсистема розпізнавання та автентифікації обличь в системі оплати »
керівник проєкту *Жданова Олена Григорівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*7*”*травня* 2020 р. №1081-с
2. Термін подання студентом проєкту “*01*”*червня* 2020 року
3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна діяльності

2. Схема структурна варіантів використань

3. Схема бази даних

4. Схема структурна класів програмного забезпечення

5. Схема структурна послідовності

6. Схема структурна компонентів програмного забезпечення

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *«13» квітня 2020 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*14.04.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*16.04.2020*	
3.	*Постановка та формалізація задачі*	*18.04.2020*	
4.	*Розробка інформаційного забезпечення*	*22.04.2020*	
5.	*Алгоритмізація задачі*	*26.04.2020*	
6.	*Обґрунтування використовуваних технічних засобів*	*27.04.2020*	
7.	*Розробка програмного забезпечення*	*29.04.2020*	
8.	*Налагодження програми*	*09.05.2020*	
9.	*Виконання графічних документів*	*10.05.2020*	
10.	*Оформлення пояснювальної записки*	*11.05.2020*	
11.	*Подання ДП на попередній захист*	*15.05.2020*	
12.	*Подання ДП на основний захист*	*01.06.2020*	
13.	*Подання ДП рецензенту*	*02.06.2020*	

Студент Іван ПОРТЯНИЙ

Керівник Олена ЖДАНОВА

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6318.00.000 ПЗ	Пояснювальна записка	68	
3	А4	ДП 6318.01.000 ТЗ	Технічне завдання	10	
4	А3	ДП 6318.02.000 ССД	Схема структурна діяльності	1	
5	А3	ДП 6318.03.000 ССВ	Схема структурна варіантів використань	1	
6	А3	ДП 6318.04.000 СБД	Схема бази даних	1	
7	А3	ДП 6318.05.000 ССК	Схема структурна класів програмного	1	
			забезпечення		
9	А3	ДП 6318.06.000 ССП	Схема структурна послідовності	1	
10	А3	ДП 6318.07.000 ССК	Схема структурна компонентів	1	
			програмного забезпечення		

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 6318 00.000.00		
Розробн.		Портяний І.С.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Жданова О.Г.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-63	
Н/контр.		Проскура С.Л.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: Підсистема розпізнавання та автентифікації обличь в системі оплати

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з п’ятьох розділів і містить 25 рисунків, 18 таблиць, 1 додаток, 20 джерел.

У дипломному проекті реалізовано підсистему розпізнавання та автентифікації обличчя в рамках системи оплати. Метою підсистеми є пришвидшення процесу оплати за рахунок спрощення та полегшення процесу підтвердження платежу. Задачами розробки є створення системи, яка дозволить реєструвати організації-користувачів та надавати їм доступ до функціоналу розпізнавання та автентифікації обличь. Користувачі матимуть можливість додавати своїх клієнтів з їхніми фотокартками, після чого зможуть виконувати автентифікацію обличчя зареєстрованих клієнтів.

У розділі загальних положень описано процес діяльності системи, наведені функціональні діаграми, проведено огляд та аналіз існуючих аналогів до системи, описано призначення, цілі та мета розробки.

Розділ інформаційного забезпечення присвячено визначенню наборів вхідних та вихідних даних, описано схеми та структури інформаційних сховищ даних.

У розділі математичного забезпечення описано процес розпізнавання та автентифікації обличчя, наведено огляд загальної архітектури та основних шарів згорткових нейронних мереж.

Розділ програмного та технічного забезпечення описує вибрані засоби розробки для даної системи, типи даних, класи, специфікації функцій, обрано та обґрунтовано архітектуру програмного забезпечення й сховищ даних.

У технологічному розділі наведено інструкцію користувача та наведено результати випробувань системи.

ПІДТВЕРДЖЕННЯ ПЛАТЕЖІВ, РОЗПІЗНАВАННЯ ОБЛИЧЧЯ, АВТЕНТИФІКАЦІЯ ОБЛИЧЧЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ПРИКЛАДНИЙ ПРОГРАМНИЙ ІНТЕРФЕЙС

ABSTRACT

The structure and scope of this paper. Explanatory note of the graduation project consists of five sections, contains 25 figures, 18 tables, 1 appendix, 20 references.

The diploma project implemented the subsystem of face recognition and authentication within the payment system. Aimed at acceleration the payment process by simplifying and facilitating the payment confirmation process. Development tasks are creating a system what will allow the registration of user companies and giving them access to the functionality of face recognition and authentication. Users will be able to add profiles with their customers' photos, and then they can authenticate the faces of their clients.

The general provisions describe the process of system activity with functional diagrams, reviews and analyzes the existing analogues, describes the purpose, goal and objectives set of development.

The information support section is dedicated to defining input and output datasets, describes schemas and structures of data repositories.

The section on mathematical support describes the process of face recognition and authentication, provides an overview of the general architecture and underlying layers of convolutional neural networks.

The software and technology section describes the main tools for development the system, the types of data, classes, function specifications, the architecture of software and data repositories are selected and grounded.

The technological part contains user guide and results of system testing.

PAYMENT CONFIRMATION, FACE RECOGNITION, FACE AUTHENTICATION, CONVOLUTIONAL NEURAL NETWORKS, WEB-API

ЗМІСТ

ВСТУП 7

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ 9

1.1 Опис предметного середовища 9

1.1.1 Опис процесу діяльності 9

1.1.2 Опис функціональної моделі 10

1.2 Огляд наявних аналогів 14

1.2.1 Alipay 14

1.2.2 FacePay24 15

1.2.3 Відмінності рішення в дипломному проекті від перелічених аналогів 16

1.3 Постановка задачі 16

1.3.1 Призначення розробки 16

1.3.2 Цілі та задачі розробки 16

Висновок до розділу 17

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ 18

2.1 Вхідні дані 18

2.2 Вихідні дані 19

2.3 Опис структури бази даних 20

Висновок до розділу 24

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ 25

3.1 Змістовна постановка задачі 25

3.2 Обґрунтування методу розв'язання 26

3.3 Опис методів розв'язання 29

3.3.1 AlexNet 29

3.3.2 VGGNet 29

3.3.3 ResNet 30

3.3.4 Inception 30

Висновок до розділу 32

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ 33

4.1 Засоби розробки 33

4.2 Вимоги до технічного забезпечення 38

4.2.1 Загальні вимоги ***38***

4.3 АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 39

4.3.1 Діаграма класів *40*

4.3.2 Діаграма послідовності *44*

4.3.3 Діаграма компонентів *45*

4.3.4 Специфікація функцій *46*

ВИСНОВОК ДО РОЗДІЛУ 48

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ **49**

5.1 КЕРІВНИЦТВО КОРИСТУВАЧА 49

5.1.1 Керівництво адміністратора *49*

5.1.1 Керівництво організації-користувача *54*

5.2 ВИПРОБУВАННЯ ПРОГРАМНОГО ПРОДУКТУ 58

5.2.1 Мета випробувань *58*

5.2.2 Загальні положення *58*

5.2.3 Результати випробувань *58*

ВИСНОВОК ДО РОЗДІЛУ 64

ЗАГАЛЬНІ ВИСНОВКИ **65**

ПЕРЕЛІК ПОСИЛАНЬ **66**

ДОДАТОК А **69**

ВСТУП

Сучасний світ постійно змінюється, у наш час відбувається стрімкий розвиток технологій та застосування їх у будь-яких сферах життя. За останні роки значних успіхів було досягнуто в сфері комп’ютерного зору. Комп'ютерний зір – теорія та технологія, яка зосереджена на проблемі допомагати комп'ютерам «бачити» та розуміти зміст цифрових зображень, таких як фотографії та відео. Однією з задач, які вирішує комп’ютерний зір, є задача розпізнавання обличчя. Розпізнавання обличчя – це проблема ідентифікації та перевірки людей на фотографії за їх обличчям. Це завдання, яке тривіально виконується людьми, навіть при різному світлі та коли обличчя змінюється з віком. Методи глибокого навчання вже зараз здатні виконувати таке ж завдання, використовуючи та вивчаючи дуже великі набори даних. З часом моделі, можливо, зможуть перевершити можливості розпізнавання обличь людини.

Зважаючи на успіхи в сфері комп’ютерного бачення, системи розпізнавання обличь впроваджуються все в більшій кількості сфер. Наприклад, пропускні системи, розблокування мобільних девайсів, пошук осіб, які знаходяться в розшуку, за допомогою камер на вулицях.

Головною ідеєю цього дипломного проекту є впровадження системи розпізнавання та автентифікації обличчя (далі – FaceAPI) в процес підтвердження оплати.

У сучасному світі спостерігається тенденція переходу від оплати готівкою до оплати карткою та до «електронних» грошей. Все більше людей перестають використовувати готівку й індустрія підлаштовується під це.

Щоб забезпечити достатній рівень безпеки при здійсненні платежів використовується підтвердження платежу наступними способами:

- введення паролю від картки;
- відправка СМС з кодом підтвердження на номер власника картки.

Дана робота присвячена методу підтвердження оплати за допомогою обличчя. Тобто для того, щоб здійснити платіж достатньо просто подивитись в камеру. Цей метод дозволяє не носити з собою картку, забути про введення паролів та навіть не турбуватись про наявність смартфону, так як не відбувається відправка коду підтвердження.

Метою даного проекту є пришвидшення та полегшення процесу проведення платежів за рахунок спрощення процесу підтвердження платежу.

Практичне значення одержаних результатів. Розроблено інструментарій, який застосовує розпізнавання та автентифікацію обличчя для задачі підтвердження платежів.

Публікації. Результати огляду варіантів вирішення задачі розпізнавання, яка реалізована в даній роботі, опубліковано в тезах доповідей на науково-практичні конференції [7].

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Є деяка організація, яка надає певні послуги для своїх клієнтів, має базу цих клієнтів та дозволяє в межах своїх послуг здійснювати платежі. Організація також повинна мати засоби для ідентифікації своїх клієнтів: пластикову картку з чіпом, QR-код, тощо.

Для прикладу організації можна взяти спортивний зал. Спортивний зал має свою систему обліку клієнтів, яка дозволяє приєднати до акаунту клієнта платіжні дані. Кожен клієнт має пластикову картку з чіпом за допомогою якої він може ідентифікувати себе в системі обліку клієнтів. Керівництво спортзалу може додати в свою систему можливість здійснювати платежі в межах свого закладу за допомогою обличчя. Для цього потрібно зареєструватись в системі FaceAPI та додати туди профілі своїх клієнтів. Щоб додати профіль клієнта достатньо вказати його унікальний ідентифікатор та додати всього три фото клієнта, який буде «прив’язаний» до вказаного профілю. Після того, як виконано вищеперераховані дій, клієнти мають можливість оплачувати послуги спортзалу використовуючи лише свій ідентифікатор в системі та за допомогою свого обличчя.

Даний дипломний проект надає можливість організація легко впровадити в свою інфраструктуру систему підтвердження платежів за допомогою обличчя, ставати організаціям більш технологічними та пропонувати своїм клієнтам зручніші умови.

1.1.1 Опис процесу діяльності

Проведемо розгляд дій, які повинні бути виконані для того, щоб використовувати систему та послідовність дій при використанні FaceAPI. Опишемо ці дії за допомогою UML-діаграми діяльності, яку наведено у графічному матеріалі, лист 1.

Перед початком використання функціоналу системи адміністратор реєструє в системі організацію та видає унікальний секретний ключ доступу, за допомогою якого організація зможе відправляти запити до системи й отримувати доступ тільки до профілів своїх клієнтів. Після того, як організація отримає змогу використовувати систему, потрібно додати профілі клієнтів – завантажити 3 фото для кожного профілю та присвоїти їм ідентифікатор. Після цих дій організація зможе надсилати запити до FaceAPI на автентифікацію своїх клієнтів, профілі яких додані в систему.

1.1.2 Опис функціональної моделі

Для опису функціональної моделі використаємо UML-діаграму варіантів використання (Use Case Diagram). Щоб побудувати діаграму спочатку потрібно визначити акторів, тобто дійових осіб системи. Після чого необхідно описати доступні в межах FaceAPI дії для кожного актору.

Визначимо наступний набір акторів:

– адміністратор системи;

– працівник IT відділу організації.

Наведемо опис кожного актору.

Адміністратор

Адміністратор має повний доступ до системи та інформаційних сховищ даних. Він керує даними зареєстрованих підприємств-користувачі та профілями їх клієнтів, видає токен доступу до системи. Також адміністратор має доступ до історії дій, які відбуваються в системі (інформація про додавання нових профілів, результати запитів на автентифікацію обличчя, тощо)

Працівник IT відділу організації

Організація-користувач реєструється в системі та отримує ключ доступу. Після чого працівник IT відділу організації додає профілі клієнтів та надсилає запити на автентифікацію до FaceAPI.

Визначимо набір функцій, який доступний для кожного типу акторів.

Доступні функції адміністратора:

- реєстрація організації в системі;
- редагування та видалення організації;
- отримання та генерування нового ключа доступу для конкретної організації-користувача;
- перегляд історії дій в системі.

Доступні функції працівника IT відділу організації:

- додавання профілю клієнта в систему;
- зміна набору фото для конкретного профілю;
- видалення профілю з системи;
- отримання головного фото для профілю;
- автентифікація обличчя клієнта організації, який має профіль в системі;
- отримання списку унікальних ідентифікаторів профілів, які створені організацією.

Тепер, коли визначено всіх акторів та функції, які їм доступні, можемо визначити дії, які актори можуть виконувати в межах системи та наведемо їх в таблиці 1.1.

На основі наведених вище варіантів використання побудовано UML-діаграму варіантів використання, що зображена у графічному матеріалі, лист 2.

Таблиця 1.1 – Дії акторів у системі

Актор	**Варіант використання**	**Опис дії варіанта використання**
Адміністратор	Реєстрація організації-користувача	Актор заповнює форму для реєстрації акаунта організації-користувача, вводить відповідні дані й отримує ключ доступу для зареєстрованого акаунту, після чого передає його організації.
	Перегляд списку зареєстрованих організацій	Актор має можливість переглядати список зареєстрованих організацій.
	Редагування даних	Актор може редагувати дані вже зареєстрованих організацій, включаючи генерацію нового ключа доступу.
	Перегляд історії дій в системі	Актор має змогу переглянути інформацію про дії, які відбуваються в системі: додавання нових профілів клієнтів, запити на автентифікацію та їх результати, тощо.
Працівник IT відділу організації	Додати профіль клієнта	Актор має можливість надіслати запит, який містить дані про ідентифікатор клієнта та фото клієнта. Після трьох завантажених фото, буде створено профіль клієнта.

Продовження таблиці 1.1

Актор	Варіант використання	Опис дії варіанта використання
	Переглянути список доданих профілів	Актору надається можливість надіслати запит до FaceAPI та отримати список ідентифікаторів клієнтів, які були додані цим актором.
	Оновити фото для існуючого профілю	Актор може надіслати запит з новими фото для профілю, щоб оновити старі фото.
	Переглянути головне фото для конкретного профілю	Актор має можливість надіслати запит з ідентифікатором профілю та отримати у відповідь основне фото профілю.
	Автентифікувати обличчя	Актор має можливість надіслати запит з ідентифікатором профілю та фото обличчя, після чого система виконає розпізнавання обличчя й порівняння з обличчям на раніше доданих фото до профілю з заданим ідентифікатором й надасть відповідь щодо автентифікації.
	Видалити профіль	Актор може надіслати запит з ідентифікатором профілю для видалення профілю, після чого всі фото, які прикріплені до нього буде видалено.

1.2 Огляд наявних аналогів

В ході процесу пошуку аналогічних рішень, не було виявлено систем, що вирішують абсолютно ідентичну задачу. Але було знайдено програмні продукти, що використовують це рішення, як підзадачу у процесі декомпозиції. Це наступні системи:

– Alipay;
– FacePay24.

Розглянемо можливості та функціонал, які надають ці системи.

1.2.1 Alipay

Alipay – платіжна система, що використовує технологію розпізнавання обличчя, яка дозволяє споживачам в Китаї оплачувати свої покупки без використання мобільного телефону та пластикової картки [8].

Використання цієї технології вирішує низку проблем:

– довготривале проходження через ворота квитків у метро у час пік (пасажири можуть пройти через ворота квитків, просто подивившись на планшети, вбудовані в автоматичні шашки, їм не потрібно брати з собою мобільний телефон);
– шахрайство у ресторанах, супермаркетах(споживачі не зможуть вийти з громадського закладу не сплативши свій рахунок, приблизно 10000 торгових точок вже використовують систему, яка дозволяє клієнтам здійснювати покупки, лише скануючи їх обличчя);
– легка оплата для людей з обмеженими можливостями(програмне забезпечення не вимагає жодної операції зі смартфоном чи розміщення пальця на сканері, що робить життя простішим для осіб з інвалідністю);
– захист рахунку(традиційним способом дуже небезпечно вводити пароль, якщо хтось стоїть поруч, система допомагає захистити свої приватні дані до рахунку).

Головним недоліком програми є конфіденційність даних. Адже існує великий ризик того, що держава може використовувати ці дані для власних цілей, таких як спостереження, моніторинг, відстеження політичних дисидентів, соціальний та інформаційний контроль, етнічне профілювання.

1.2.2 FacePay24

FacePay24 – система біометричної оплати покупок для роздрібних торгових мереж за підтримки платіжною системи Visa та ПриватБанку [9]. Технологія не є широковикористовуваною й використовується лише в чотирьох магазинах національної мережі HOP HEY.

Головними перевагами FacePay24 є:

- іноваційний спосіб зробити шопінг зручніше, перехід на безготівкові розрахунки (українцям не знадобиться носити з собою ні гаманець, ні смартфон, ні платіжну карту);
- безпека здійснення сплат (біометрія відкриває нові можливості з точки зору аутентифікації і безпосередньо здійснення платежів, гарантує їх безпеку, і в подальшому буде в значній мірі впливати на розвиток електронних платежів)
- конфіденційність (FacePay24 використовує одну з провідних в світі систем автоматичного розпізнавання осіб Amazon Rekognition і дає можливість масштабувати цю послугу майже на будь-яку торговельну точку. Amazon Recognition створена на базі безпечної, надійної і перевіреної технології глибокого навчання, розробленої експертами Amazon в області комп'ютерного зору для щоденного аналізу мільярдів зображень і відео);

Система не використовує особисту інформацію, яка може міститися в вашому контенті, для орієнтування продуктів, послуг або маркетингу.

1.2.3 Відмінності рішення в дипломному проекті від перелічених аналогів

Ще кілька років тому система оплати обличчям - була фантастикою, але сьогодні - це стало реальністю. Порівнюючи доступні аналоги з моїм рішенням, можна побачити деякі відмінності. Розроблена мною технологія працює швидше, бо вона потребує підтвердження особистості, а не пошуку особи в усій базі користувачів. Також це дає перевагу в зменшенні похибки у результаті, тому що програма порівнює зображення особи та три інших фотографії користувача у базі, під яким хоче автентифікуватись людина.

Не можна не сказати про широке використання для будь-якої компанії – від малого бізнесу до великих корпорацій, бо для цього потрібна лише система обліку своїх клієнтів та можливість додати до профілю клієнта платіжні дані.

1.3 Постановка задачі

1.3.1 Призначення розробки

Призначенням Системи FaceAPI є розпізнавання та автентифікація обличчя для підтвердження платежів у системі оплати.

1.3.2 Цілі та задачі розробки

Цілями розробки Системи FaceAPI є покращити процес оплати за рахунок:

- спрощення процесу підтвердження платежу;
- підвищення рівня безпеки під час проведення платежів.

Для того, щоб досягнути поставлених цілей необхідно реалізувати наступні задачі:

- створення, редагування та видалення організацій-користувачів, які використовуватимуть систему;

- створення, редагування та видалення клієнтів організації для розпізнавання обличчя;
- розпізнавання та автентифікація обличчя по фото.

Висновок до розділу

У рамках цього розділу проведено аналіз предметної області та описано процес використання системи, яка розроблена в даному дипломному проекті. Наведено опис процесу діяльності, розроблено функціональну модель, виділено групи користувачів та функції, які може виконувати кожна група.

Виконано пошук та дослідження аналогів, в ході якого було визначено, що ідентичних аналогів немає, але існую близькі по функціоналу системи. Здійснено порівняння FaceAPI з цими системами.

Враховуючи опис предметного середовища та визначені функціональні вимоги було сформульована призначення, мету й задачі розробки.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Вхідними дані можуть надходити в систему FaceAPI з різних джерел, а саме від:

– адміністратора;

– організації-користувача.

Розглянемо детальніше дані, які надходять від кожного з вищезгаданих джерел, та їх формат.

Дані від адміністратора

Для входу в систему адміністратор повинен пройти авторизацію. Для цього потрібно ввести логін та пароль.

Для того, щоб зареєструвати в системі нову організацію-користувача адміністратор повинен ввести наступні дані про організацію:

– назва;

– електронна пошта;

– додаткова інформація (за необхідністю).

Адміністратор має можливість змінити вищезгадані дані при необхідності.

Дані від організації-користувача

Користувач системи має можливість:

– створювати, редагувати та видаляти профілі своїх клієнтів;

– отримувати список усіх зареєстрованих профілів клієнтів;

– проводити автентифікацію обличь.

Робота з системою відбувається за допомогою HTTP-запитів [6]. Розглянемо які саме дані потрібно передавати в запитах для вищеописаних дій.

Для виконання будь-яких дій з системою, організація-користувач повинна завжди передавати в запиті наступні дані:

– унікальний ідентифікатор організації в системі;
– ключ доступу до системи.

Вищезгадані дані організація отримує після реєстрації. Також цих даних достатньо для того, щоб отримати список усіх зареєстрованих профілів клієнтів.

Для реєстрації, редагування та авторизації профілів клієнтів у системі потрібно додатково надсилати в запиті:

– унікальний ідентифікатор клієнта;
– зображення з обличчям у форматі .jpg або .jpeg.

Для видалення клієнта достатньо додатково надіслати унікальний ідентифікатор клієнта.

2.2 Вихідні дані

Вихідні дані можна розділити на дві групи:

– для адміністратора;
– для організації-користувача.

Вихідні дані для адміністратора

Після того, як адміністратор зареєстрував організацію, система повертає наступні дані:

– унікальний ідентифікатор організації;
– згенерований ключ доступу до системи для зареєстрованої організації.

Адміністратор також має можливість згенерувати новий ключ доступу для конкретної організації. У цьому випадку вихідними даними буде новий ключ доступу.

Адміністратор може переглянути історію подій у системі. Кожна подія містить в собі такі вихідні дані:

– ідентифікатор компанії;
– ідентифікатор клієнта;
– опис дії.

Вихідні дані для організації-користувача

Вихідними даними для організації користувача є:

– список усіх зареєстрованих профілів клієнтів;
– результат автентифікації.

Список зареєстрованих клієнтів містить інформацію про кожного клієнта з наступними даними:

– ідентифікатор клієнта;
– ідентифікатор компанії;
– статус;
– фото клієнта;
– список ідентифікаторів обличчя клієнта в колекції Amazon Rekognition.

Результатом автентифікації є наступні дані:

– HTTP-код результату запиту;
– Повідомлення про те, що обличчя автентифіковано або не автентифіковано.

2.3 Опис структури бази даних

Для розробки системи було використано нереляційну базу даних MongoDB для зберігання даних про користувачів та їх клієнтів, а також сховище даних Amazon S3 для зберігання зображень.

Так як MongoDB документо-орієнтована база даних, то замість таблиць і записів з атрибутами будуть розглядатись колекції та документи з полями. У

розробленій моделі бази даних створено 3 колекції. Наведемо перелік колекцій та сутностей, що їм відповідають у таблиці 2.1.

Таблиця 2.1 – Перелік колекції та сутностей

№	Назва колекції	Сутність
1	Company	Організація
2	Profile	Профіль клієнта
3	Action	Дія в системі

Колекція Company містить дані про організацію-користувача, Profile – про клієнтів організації, а колекція Action потрібна для того, щоб зберігати список дій, які відбуваються в системі.

У таблиці 2.2. детальний опис кожної колекції.

Таблиця 2.2 – Детальний опис таблиць бази даних

Назва колекції	Назва поля документу	Тип даних	Опис поля
Company	_id	ObjectId	Первинний ключ
	name	String	Назва організації
	email	String	Електронна пошта
	token	String	Ключ доступу
	information	String	Інформація про організацію
Profile	_id	ObjectId	Первинний ключ
	profile_id	String	Ідентифікатор клієнта в системі
	company_id	String	Ідентифікатор організації в системі
	photos	Array	Список фото клієнта
	main_photo	Object	Головне фото клієнта

Продовження таблиці 2.2

Назва колекції	Назва поля документу	Тип даних	Опис поля
	active	String	Статус активності
Action	_id	ObjectId	Первинний ключ
	company_id	String	Ідентифікатор організації в системі
	profile_id	String	Ідентифікатор клієнта в системі
	action	String	Дія в системі
	datetime	Datetime	Час виконання дії

Колекція Profile містить дані про фото клієнта у полях photos та main_photo. Ці дані мають тип Object, поля яких описано в таблиці 2.3.

Таблиця 2.3 – Опис даних про фото клієнта

Назва поля	Тип даних	Опис поля
face_id	String	Ідентифікатор обличчя в системі Amazon Rekognition
url	String	Дані про розміщення зображення в сховищі даних Amazon S3

На основі описаної вище структури бази даних побудовано ER-діаграму, що зображена у графічному матеріалі, лист 3.

Окрім бази даних MongoDB, система використовує сховище даних Amazon S3 для збереження отриманих зображень. У таблиці 2.4 наведено список контейнерів(buckets) та їх призначення.

Таблиця 2.4 – Опис сховища Amazon S3

Назва контейнеру	**Опис контейнеру**
faceapi.profiles	Контейнер для зберігання зображень обличь зареєстрованих клієнтів
faceapi.auth	Контейнер для зберігання зображень обличь клієнтів, які проходять автентифікацію

У кожному контейнері зображення має свій формат назви. Для контейнеру faceapi.profiles назва файлу зображення містить наступний формат:

<company_id>_<profile_id>_photo_< photo_number>,

де *company_id* – ідентифікатор компанії, яка додала фото;

profile_id – ідентифікатор клієнта;

photo_number – порядковий номер фото клієнта.

Приклад назви зображення наведено на рисунку 2.1.

Рисунок 2.1 – Приклад назви зображення для контейнеру faceapi.profiles

Для контейнеру faceapi.auth назва файлу зображення містить наступний формат:

<company_id>_<profile_id>_auth_photo_<current_datetime>,

де *company_id* – ідентифікатор компанії, яка додала фото;

profile_id – ідентифікатор клієнта;

current_datetime – поточна дата та час.

Приклад назви зображення наведено на рисунку 2.2.

Рисунок 2.2 – Приклад назви зображення для контейнеру faceapi.auth

Висновок до розділу

У цьому розділі було сформовано список всіх вхідних та вихідних даних для адміністратора так організації-користувача системи. Також було наведено інформацію щодо баз даних, які використовувались при розробці: опис колекцій та документів MongoDB і опис контейнерів та формату збереження зображень в Amazon S3.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

У рамках цього дипломного проекту розв’язуються задачі розпізнавання та автентифікації обличчя. Розпізнавати обличчя потрібно при завантаженні фото пів час запиту на реєстрацію профілю клієнта та при завантаженні фото під час запиту на автентифікацію. Автентифікація обличчя відбується після запиту на автентифікацію й полягає в порівнянні обличчя клієнта(якого потрібно автентифікувати) та обличчя з фото. Розглянемо методи розв’язання цих задач.

3.1 Змістовна постановка задачі

Розпізнавання обличчя це задача, яка вирішується за допомогою засобів комп’ютерного зору. Комп'ютерний зір – теорія та технологія, яка зосереджена на проблемі допомоги комп'ютерам «бачити» та розуміти зміст цифрових зображень, таких як фотографії та відео.

Розпізнавання обличчя – це проблема ідентифікації та перевірки людей на фото за їх обличчям. Це завдання, яке тривіально виконується людьми, навіть при різному світлі, при змінах обличчя з віком або при наявності аксесуарів чи волосся на обличчі. Але за допомогою методів глибокого навчання та використанні дуже великих наборів даних є можливість створювати моделі нейронних мереж, які можуть постійно навчатися, а з часом, можливо, перевершити можливості розпізнавання обличь людини.

Розпізнавання обличчя часто описується як процес, який включає в себе чотири етапи: виявлення обличчя, вирівнювання обличчя, визначення характеристик і розпізнавання обличчя [10]. Розглянемо ці етапи детальніше:

– виявлення обличчя (Face Detection) – пошук одного або більше обличь на зображенні та виділення їх розміщення;

– вирівнювання обличчя (Face Alignment) – нормалізація обличчя, щоб воно відповідало структурі інших обличь в базі даних, наприклад геометрії та фотометрії;
– визначення характеристик (Feature Extraction) – отримання характеристик обличчя, які можна використовувати для завдання розпізнавання;
– розпізнавання обличчя (Feature Matching / Face Recognition) – знаходження відповідності обличчя одному або більше відомим обличчям у підготовленій базі даних.

Схема вищеописаного процесу зображена на рисунку 3.1.

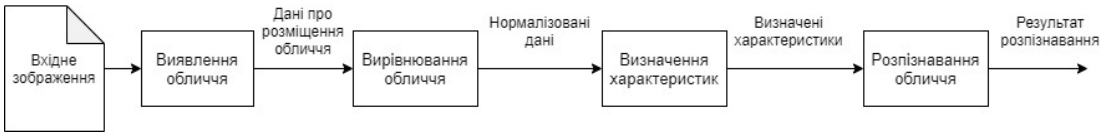


Рисунок 3.1 – Схема процесу розпізнавання обличчя

Отже, система FaceAPI отримує в запиті зображення з обличчям клієнта. Спочатку виконується ряд перевірок зображення: на наявність обличчя, на розмір обличчя на фото, а також перевіряється чи відкриті очі на фото. Після цього відбувається вирівнювання обличчя та визначення ознак, які порівнюються з уже визначеними ознаками раніше завантажених в базу обличь.

3.2 Обґрунтування методу розв’язання

Задачу розпізнавання обличчя можна розв’язувати за допомогою методів неглибокого (shallow learning) [3-5] та глибокого навчання (deep learning) [1]. Алгоритми глибокого навчання показують себе набагато краще й дозволяють отримати більш широкий набір характеристик обличчя. На відміну від методів неглибокого навчання, для методів глибокого навчання не є проблемою справлятись з розпізнаванням обличчя при маскуванні, зміні пози чи фону, а також при різному виразу обличчя [7].

У глибокому навчанні є різні підходи, такі як згорткові нейронні мережі (Convolutional Neural Networks – CNN), автоенкодери (Stacked Autoencoder) [11] або глибинні мережі переконань (Deep Belief Network) [12]. Але основним інструментом для класифікації і розпізнавання обличь на фото є CNN [7].

Згорткова нейронна мережа за рахунок застосування спеціальної операції – згортки – дозволяє зменшити кількість інформації в пам’яті (зменшити розмірність даних) й за рахунок цього краще справляється з картинками більш високого розширення. Спочатку відбувається виділення опорних фрагментів, таких як ребра, контури або межі. Далі відбувається обробка цих фрагментів, щоб виявити повторювані фрагменти – текстури, з яких потім визначаються об’єкти на фото.

По суті кожен шар нейронної мережі використовує власне перетворення. Якщо на перших шарах мережа оперує такими поняттями, як «ребра», «межі», то далі використовуються поняття «текстури», «частини об'єктів». У результаті такої обробки ми можемо правильно класифікувати картинку або отримати на кінцевому кроці певний об'єкт на фото.

На рисунку 3.2 зображено типову архітектуру згорткової нейронної мережі. Вона містить наступні базові шари: Convolutional Layer, Pooling Layer, ReLU(Rectified Linear Unit) Layer та Fully Connected Layer [13].

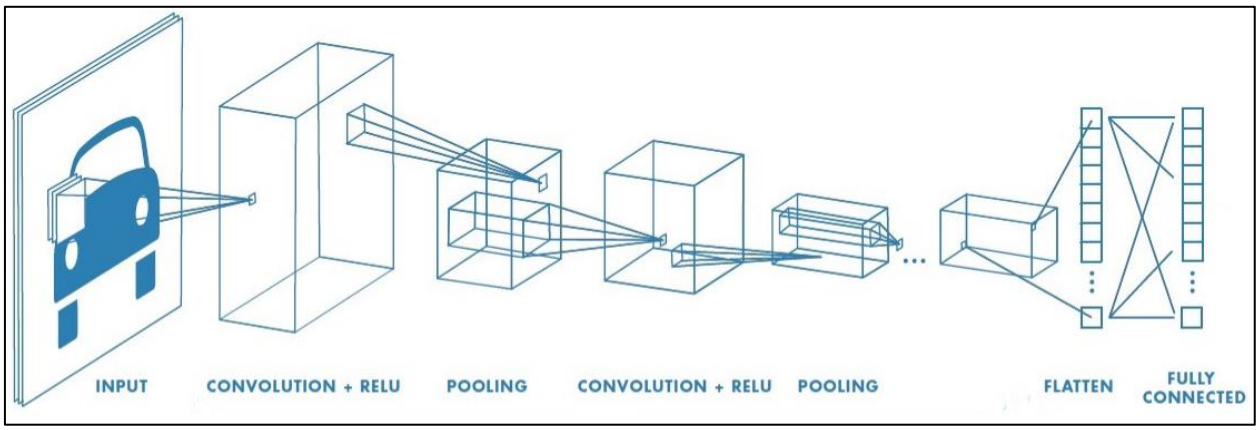


Рисунок 3.2 – Типова архітектура згорткової нейронної мережі

Згортковий шар (Convolutional layer) є одним з основних шарів згорткової мережі, який виконує основну масу обчислювальної роботи.

Головне призначення цього шару – це витягнути ознаки з вхідних даних, у нашому випадку вхідними даними є картинки. За допомогою згортки зберігаються просторові пропорції між пікселями та визначаються особливості зображення використовуючи маленькі квадрати вхідного зображення. Вхідне зображення «згортається» завдяки застосуванню фільтрів до його частин, за допомогою чого розмірність картинки зменшується й створюється карта ознак (активаційна карта) вихідного зображення, після чого вона передається наступному згортковому шару [7].

Pooling layer зменшує розмірність кожної активаційної карти, але зберігає найбільш важливу інформацію. Вхідне зображення ділиться на набір прямокутників, які не перетинаються між собою, після чого до кожного з них застосовується понижуюча дискретизація за допомогою нелінійної операції, наприклад знаходження середнього (average pooling) або максимального (max pooling). У цьому шарі забезпечується найкраще узагальнення та найшвидша збіжність. Цей шар стійкий до спотворень й зазвичай розміщується між згортковими шарами [7].

ReLU являється нелінійною операцією – це поелементна операція, що означає, що вона застосовується до кожного елементу активаційної матриці. Застосування ReLU замінює всі від'ємні значення в активаційній карті на нуль. Ми можемо визначити цю операцію як $f(x)$, яка на вхід отримує значення елементу активаційної матриці x і визначається наступним чином:

$$f(x) = \max(0, x),$$

де x – значення елементу активаційної матриці [7].

Останнім шаром є Fully Connected Layer (FCL). Термін Fully Connected Layer означає, що кожен фільтр на попередньому рівні пов'язаний з кожним фільтром на наступному рівні. Цей шар отримує оброблені попередніми шарами ознаки картинки, та на основі отриманих даних класифікує вхідне зображення по різних класах на основі навчального набору даних [7].

3.3 Опис методів розв'язання

Існує багато видів архітектури даних нейронних мереж, найпопулярніші з них: AlexNet, VGGNet, ResNet та Inception. Розглянемо кожну з цих архітектур [7].

3.3.1 AlexNet

AlexNet було створено з потреби покращити результати нейронної мережі ImageNet. Це одна з перших глибоких згорткових мереж, яка досягла значної точності з значенням 84,7% порівняно з другим найкращим результатом з точністю 73,8% [7].

Нейронна мережа складається з наступних шарів: 11x11, 5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum. Після кожного convolution і fully-connected шару додається активація ReLU. Мережа налічує 62 мільйони параметрів [7]. На рисунку 3.3 наведено архітектуру даної нейронної мережі [14].

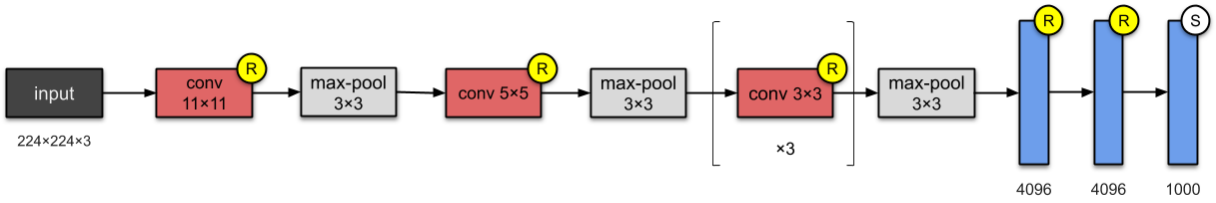


Рисунок 3.3 – Архітектура нейронної мережі AlexNet

3.3.2 VGGNet

VGGNet було створено задля зменшення часу навчання моделі та з необхідності зменшення кількості параметрів у convolution-шарах.

Існує кілька варіантів VGGNet (VGG16, VGG19 тощо), які відрізняються лише загальною кількістю шарів у мережі. Детальніше розглянемо VGG16 [7].

VGG16 має 13 convolutional і 3 fully-connected шарів, включаючи ReLU. Ця мережа використовує фільтри меншого розміру (2 × 2 і 3 × 3), ніж AlexNet.

VGG16 має загалом 138 мільйонів параметрів [7]. На рисунку 3.4 наведено архітектуру даної нейронної мережі [14].

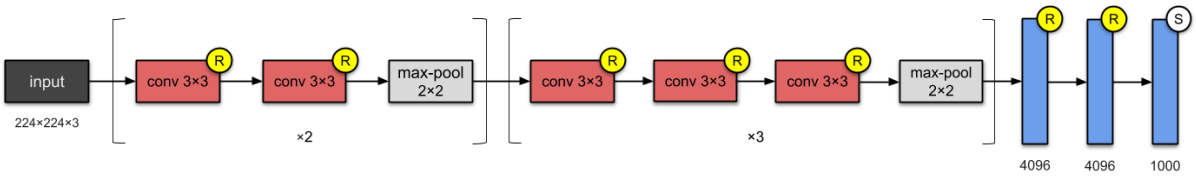


Рисунок 3.4 – Архітектура нейронної мережі VGGNet

3.3.3 ResNet

ResNet має близько 11 мільйонів параметрів. Архітектура складається з convolutional шарів з фільтрами розміром 3x3 (так само, як VGGNet), має лише 2 pooling шари на початку та вкінці мережі [7]. На рисунку 3.5 наведено архітектуру даної нейронної мережі [14].

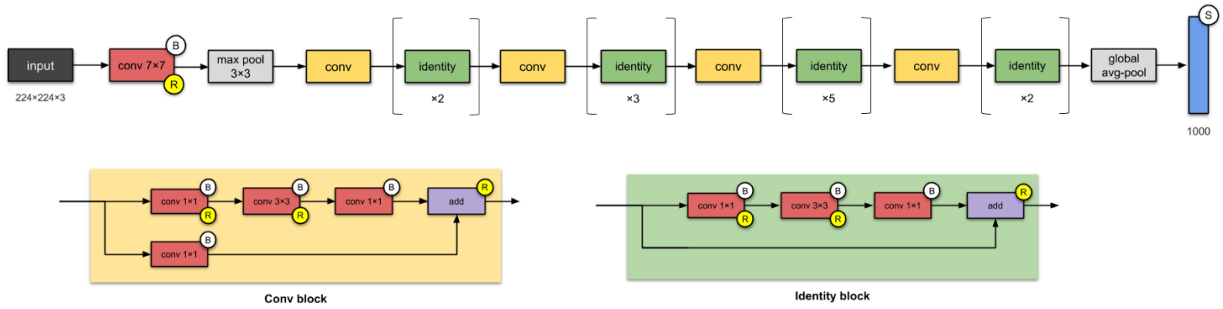


Рисунок 3.5 – Архітектура нейронної мережі ResNet

3.3.4 Inception

Inception збільшує простір нашої мережі, з якої ми можемо вибрати найкращу мережу після етапу навчання моделі. Замість того, щоб просто заглиблюватися за кількістю шарів, відбувається збільшення в ширину, тобто в межах одного шару реалізовано кілька ядер різного розміру [7]. На рисунку 3.6 наведено архітектуру даної нейронної мережі [14].

Рисунок 3.6 – Архітектура нейронної мережі Inception

Розглянемо таблицю 3.1, в якій наведено точності вищезгаданих архітектур.

Таблиця 3.1 – Точність описаних архітектур

Архітектура	**Кількість параметрів**	**Точність**
AlexNet	62 млн	84.70%
VGGNet	138 млн	92.30%
Inception	6.4 млн	93.30%
ResNet-152	60.3 млн	95.51%

З таблиці бачимо, що найкращу точність має ResNet-152, але для задачі підтвердження платежів цієї точності недостатньо. Тому для забезпечення безпеки проведення підтвердження платежів за допомогою обличчя в даному дипломному проекті було обрано нейронну мережу, точність якої досягає позначки в 99%. Це нейронна мережа, яка розроблена в Amazon. Доступ до неї здійснюється через сервіс Amazon Rekognition [15].

Висновок до розділу

У цьому розділі дипломного проекту була сформульована постановка задачі розпізнавання обличчя та розглянуто методи, якими можна розв’язати дану задачу.

Найкраще з задачею розпізнавання обличчя справляються згорткові нейронні мережі. У розділі наведено опис згорткових нейронних мереж та їх архітектури. Було розглянуто найпопулярніші архітектури та наведено їх короткий опис.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для того, щоб розробити програмний продукт було використано наступні технології та засоби розробки:

- асинхронне програмування;
- мова програмування Python та середовище розробки PyCharm;
- фреймворк для асинхронного програмування aiohttp;
- сервіс Amazon Rekognition та boto3 – бібліотека для роботи з сервісом;
- сервіс Amazon S3 для зберігання фото;
- база даних MongoDB та umongo – бібліотека для роботи з базою даних;
- технологія керування версіями git.

Розглянемо ці технології та засоби нижче та обґрунтуємо вибір.

Асинхронне програмування

Зазвичай програмний код виконується послідовно, тобто кожна операція в коді виконується одна за одною. Якщо одна з функцій залежить від результату виконання іншої, то вона повинна чекати доки потрібна їй функція завершить виконання й поверне результат. З точки зору користувача в такі моменти виконання програми буде призупинено. Така поведінка говорить про неправильне використання процесорного часу. Цю проблему можна вирішити за допомогою асинхронного програмування, за допомогою якого можна досягнути підвищення продуктивності програми й підвищення швидкості відповідей від програми [16].

Асинхронне програмування це один з типів паралельного програмування. Особливістю асинхронного програмування є той момент, що одиниця роботи програми може виконуватись окремо від основного потоку

програми, а коли виконання роботи завершено, то відбувається сповіщення основного потоку про завершення або збій під час роботи. На рисунку 4.1 можемо побачити різницю у виконанні синхронної та асинхронної програми.

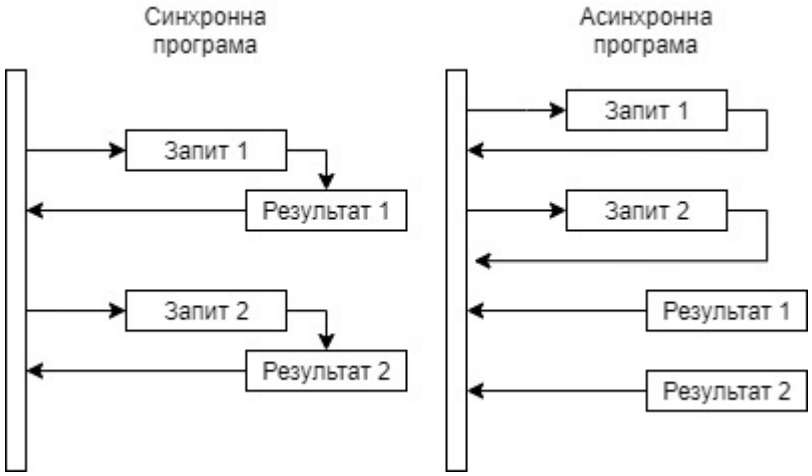


Рисунок 4.1 – Архітектура нейронної мережі Inception

В останні роки багато уваги приділяється асинхронному програмуванню. Хоч цей підхід є складнішим за традиційний синхронний варіант, але водночас він є набагато ефективнішим. Особливо ефективним є використання асинхронно програмування для обробки HTTP-запитів. Так як взаємодія з розроблюваним в дипломному проекті програмному продукті відбується через HTTP-запити, то вибір саме асинхронної технології програмування дуже добре підходить.

Python

Python – це потужна інтерпретована мова програмування з відкритим вихідним кодом, яка підтримує багато парадигм програмування. Python оптимізований для забезпечення високої продуктивності програмістів, легкого розуміння коду, якості програмного забезпечення й водночас є дуже легким для початку роботи з ним. Синтаксис мови Python максимально спрощений, що дозволяє вивчити її за відносно короткий час [2].

Python підтримує широкий набір стилів програмування, є зручним для роботи з об’єктно-орієнтованим програмуванням і функціональним програмуванням.

Python це інтерпретована мова, тобто переведення коду на машинний відбувається прямо під час виконання програми, на відміну від компільованих мов, де компілятор спочатку переводить написаний код в машинний, а далі запускає його на виконання. Окрім цього Python підтримує майже всі операційні системи й може працювати як на портативних комп'ютерах, так і на великих серверах.

Ще однією перевагою Python є підтримка модулів та пакетів, що дозволяє будувати модульний програмний продукт та повторно використовувати вже написаний код.

Єдиним значним недоліком є швидкість виконання, вона не завжди є така швидка як у низькорівневих мовах C і C++. Така проблема виникає при оперуванні великими розмірами даних, що не є проблемою для програмного продукту, який розробляється в межах дипломного проекту.

Одною з головних причин вибору Python для цього дипломного проекту є його підтримка асинхронного програмування і вбудований з версії 3.5 синтаксис, який дозволяє легко писати асинхронний код.

У якості середовища розробки було обрано PyCharm, так як він є дуже зручним для розробки, має багато вбудованих підказок та перевірок синтаксису, а також плагінів, які полегшують та пришвидшуються розробку.

aiohttp

Для Python існує багато фреймворків для веб-розробки: Django, Flask, Pylons. Але вони не підтримують асинхронне програмування. Відносно нещодавно з'явився новий фреймворк – aiohttp – який створено саме для розробки асинхронних веб-застосунків. Цей фреймворк було створено самими розробниками мови Python і він позиціонується як концепт головного Python-фреймворку для вебу.

Основою aiohttp є нескінченний цикл, в якому очікують на задачі корутини (об'єкти, які не блокують введення та виведення даних). Доки в корутині не відбудуться всі обрахунки вона «засинає», а інтерпретатор може

обробляти інші корутини. Часто затримки серверу відбуваються саме за рахунок очікування відповіді від бази даних і доки ця відповідь не повернеться й не обробиться, всі інші запити чекатимуть своєї черги. У такому випадку з aiohttp інші запити будуть також оброблюватись, доки очікуватиметься відповідь від бази даних.

Окрім вищезгаданих плюсів, aiohttp може використовувати для написання як серверу, так і клієнту веб-застосунку.

Так як розроблюваний у даному дипломному проекті програмний продукт має клієнт-серверну архітектуру, то aiohttp буде використовуватись і для написання API для серверу, і для написання клієнта.

Amazon Rekognition

У дипломному проекті використовується механізм розпізнавання та автентифікації обличчя. Для реалізації цього механізму було застосовано сервіс Amazon Rekognition, який розроблено вченими з Amazon, які спеціалізуються на комп'ютерному зорі [15]. Цей сервіс реалізує функції аналізу зображень та відео за допомогою перевіреної й високо-масштабованої технології глибокого машинного навчання.

Amazon Rekognition може ідентифікувати об'єкти, людей, текст, сцени і конкретні дії на зображеннях чи відео. Сервіс з високою точністю аналізує обличчя і відкриває широкі можливості для розпізнавання, аналізу та порівнювання обличь в різних прикладах використання, де потребується перевірка користувачів, підрахунок кількості людей або ж забезпечення громадської безпеки. Amazon Rekognition включає в себе простий та зручний API, через який можна швидко аналізувати будь-які зображення чи відео, які зберігаються в Amazon S3, який буде описано далі. Rekognition постійно навчається на нових даних й сервір розпізнавання обличь постійно покращується.

Сервіс Amazon Rekognition має дуже високу точність, це підтверджується тим, що сервіс використовують дуже багато великих

компаній (наприклад, Influential, Marinus Analytics, CBS) та навіть правоохоронні органи США, такі як імміграційна та митна поліції. Через свою високу точність сервіс добре підходить для реалізації задачі розпізнавання та автентифікації обличчя в дипломному проекті.

Amazon S3

Amazon S3 (Amazon Simple Storage Service) – це сервіс для зберігання об’єктів, який має високі показники продуктивності, масштабованості, доступності та безпеки даних (забезпечення надійності збереження даних на рівні 99.9%) [17].

Перевагами цього сервісу є:

- можливість легко збільшувати і зменшувати ресурси сховища у відповідності з коливаннями потреб;
- забезпечення надійності даних, завдяки автоматичного створення копій всіх об’єктів, що зберігаються;
- захист даних від несанкціонованого доступу за допомогою інструментів шифрування та керування доступом;
- простота використання за допомогою API.

Плюсом зберігання зображень саме в S3 є інтеграція даного сервісу з Amazon Rekognition, що дозволяє значно пришвидшити роботу з зображеннями.

boto3

Для роботи з вищеописаними сервісами через Python використовується бібліотека boto3, яка дозволяє розробникам Python створювати, налаштовувати і керувати сервісами Amazon. Бібліотека надає простий у використанні об’єктно-орієнтований API та низькорівневий доступ до сервісів Amazon.

MongoDB

У якості бази даних в дипломному проекті було обрано MongoDB. Це нереліаційна, крос-платформна, документо-орієнтована база даних, вона забезпечує високу продуктивність і масштабованість. На відміну від реляційних баз даних, де оперується поняття таблиць та записів, MongoDB використовує концепцію колекцій та документів [18].

Колекція – це група документів, яка є еквівалентом для таблиць у реляційних базах. Всі документи в колекції можуть зберігати складну структуру даних. Документ можна показати як сховище ключів та значень.

Для роботи з базою даних використовується бібліотека umongo та асинхронний клієнт для роботи з БД AsyncIOMotorClient. umongo – це ORM (Object-Relational Mapping) для Python та MongoDB, яка дозволяє зв’язувати бази даних з концепцією об’єктно-орієнтованого програмування. Тобто для кожної колекції з бази даних буде створено клас, з яким набагато зручніше працювати, ніж з документами напряму.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Для надання можливості запуску та функціонування Системи FaceAPI необхідно забезпечити наявність серверу з наступними вимогами:

- оперативна пам’ять – не менше 2 Гб;
- жорсткий диск – 50 Гб;
- процесор – чотирьохядерний процесор з частотою 2.1 Гц;
- операційна система Ubuntu 16.04;
- встановлений Docker.

Для можливості працювати з Системою FaceAPI (реєструвати нові профілі клієнтів, проводити автентифікацію обличчя) потрібно мати доступ до мережі інтернет та мати можливість відправляти HTTP-запити.

4.3 Архітектура програмного забезпечення

Система FaceAPI має клієнт-серверну архітектуру. Як клієнт, так і сервер розроблено мовою програмування Python і фреймворком aiohttp. Сервер має зовнішні з'єднання з базою даних MongoDB та сервісами Amazon S3 і Amazon Rekognition. На стороні серверу реалізовано REST API, що надає можливість організаціям використовувати FaceAPI.

Клієнт-серверна архітектура це обчислювальна модель, в якій сервер розміщує та керує більшістю ресурсів та послуг, які використовує клієнт [19]. У цьому типі архітектури розглядається один або декілька клієнтських ком'ютерів, які підключені до серверу через мережеве або інтернет з'єднання.

Перевагами цієї архітектури є:

- відсутність дублювання коду з серверу на клієнті;
- знижені вимоги до клієнтських комп'ютерів, так як всі обрахунки проводяться сервером;
- збереження даних на сервері, який у більшості випадків більш захищений.

Одним з недоліків обраної архітектури є те, що коли на сервері відбувається якийсь збій, який приводить його в неробочий стан, то вся обчислювальна мережа стає недоступною.

Для того щоб організація-користувач мала змогу працювати з Системою FaceAPI було розроблено REST(**RE**presentational **S**tate **T**ransfer) API [20]. REST API – це архітектурний стиль для розподілених систем, який слідує наступним принципам:

а) клієнт-сервер – відокремлення користувацького інтерфейсу від обчислень та збереження даних на сервері;

б) не збереження стану – кожен запит не повинен використовувати будь-який стан, збережений на сервері, а повинен містити всю потрібну інформацію в самому запиті;

в) кешованість – сервер має можливість зберігати дані в кеш та використовувати їх для нових запитів;

г) уніфікований інтерфейс – застосування принципу загальності розробки програмного забезпечення;

д) багаторівнева система – архітектура складається з багатьох шарів.

4.3.1 Діаграма класів

Під час написання Системи FaceAPI використовувались об’єктно-орієнтована та функціональна парадигми програмування.

Об’єктно-орієнтована парадигма програмування використовується для написання класів, які реалізують моделі об’єктів з бази даних, а також для класів, які реалізують валідацію даних, отриманих з запитів API.

Функціональна парадигма програмування використовується для:

- написання функцій, які реалізують REST API;
- реалізації бізнес логіки програмного продукту;
- реалізації функціоналу роботи з сервісами Amazon S3 та Amazon Rekognition.

Опис специфікації функцій наведено у пункті 4.3.4 пояснювальної записки.

Перелік створених класів та їх опис наведено у таблиці 4.1.

Таблиця 4.1 – Опис класів

Клас	Опис
Company	Клас, який описує модель даних Організація й реалізує функціонал роботи з об’єктом бази даних.
Profile	Клас, який описує модель даних Профіль клієнта й реалізує функціонал роботи з об’єктом бази даних.
Photo	Клас, який описує модель даних Фото й реалізує функціонал роботи з об’єктом бази даних.

Продовження таблиці 4.1

Клас	Опис
Action	Клас, який описує модель даних Дія в системі й реалізує функціонал роботи з об’єктом бази даних.
ProfileContext	Клас який описує контекст HTTP-запитів при роботі з профілями клієнтів.
CompanySchema	Клас, який реалізує валідацію даних про організацію з HTTP-запитів при роботі з профілями організацій.
ProfileSchema	Клас, який реалізує валідацію даних про клієнтів з HTTP-запитів при роботі з профілями клієнтів.
_Photo	Клас, який реалізує валідацію даних про фото з HTTP-запитів при роботі з профілями клієнтів.
FaceAPIManager	Клас, який реалізує функціонал з’єднання та роботи клієнту з сервером.

Зв’язок між класами показано на UML-діаграмі класів, яку наведено у графічному матеріалі, лист 4.

У таблиці 4.2 наведено опис атрибутів класів(окрім класів для валідації даних).

Таблиця 4.2 – Опис атрибутів класів

Клас	Атрибут	Тип	Опис
Company	id	ObjectId	Ідентифікатор запису в базі даних
	name	str	Назва організації
	email	str	Електронна пошта
	token	str	Ключ доступу

Продовження таблиці 4.2

Клас	Атрибут	Тип	Опис
	information	str	Інформація про організацію
Profile	id	ObjectId	Ідентифікатор запису в базі даних
	profile_id	str	Ідентифікатор клієнта в системі
	company_id	str	Ідентифікатор організації в системі
	photos	list	Список фото клієнта
	main_photo	Photo	Головне фото клієнта
	active	str	Статус активності
Photo	face_id	String	Ідентифікатор обличчя в системі Amazon Rekognition
	url	String	Дані про розміщення зображення в сховищі даних Amazon S3
Action	id	ObjectId	Ідентифікатор запису в базі даних
	company_id	String	Ідентифікатор організації в системі
	profile_id	String	Ідентифікатор клієнта в системі
	datetime	Datetime	Час виконання дії

У таблиці 4.3 наведено опис методів класів(окрім класів для валідації даних).

Таблиця 4.3 – Опис методів класів

Клас	**Метод**	**Опис**
Company	regenerate_token()	Генерація нового ключа доступу для організації
	is_activated()	Перевірка чи профіль організації активований
	activate()	Активація профілю організації
	deactivate()	Деактивація профілю організації
Profile	activate()	Активація профілю клієнта
	deactivate()	Деактивація профілю клієнта
FaceAPI Manager	get_company_list()	Запит на отримання списку всіх зареєстрованих організацій
	get_company(doc_id)	Запит на отримання даних про організацію
	add_company(company_data)	Запит на додавання нової організації
	edit_company(doc_id, company_data)	Запит на редагування даних організації
	delete_company(doc_id)	Запит на видалення організації
	regenerate_company_token(doc_id)	Запит на генерацію нового ключа доступу для організації
	register_profile(company_id, profile_id, token, photo)	Запит на реєстрацію нового профілю клієнта
	auth_profile(company_id, profile_id, token, photo)	Запит на автентифікацію клієнта

4.3.2 Діаграма послідовності

Розглянемо перелік дій, які виконуються при запиті від організації-користувача на автентифікацію обличчя клієнта.

Система FaceAPI отримує HTTP-запит на автентифікацію обличчя від організації користувача, який містить наступні дані:

- унікальний ідентифікатор організації в системі;
- ключ доступу;
- унікальний ідентифікатор клієнта;
- файл зображення з обличчям клієнта.

У першу чергу відбувається перевірка доступу до FaceAPI, а саме перевіряється чи дійсно отриманий ключ доступу відповідає ключу доступу даної організації. У разі невідповідності ключів доступу буде надіслано відповідь з відповідним повідомленням. Якщо доступ підтверджено, то відбувається перевірка наявності профілю клієнта з таким ідентифікатором, а також валідація розміру та формату отриманого файлу. У разі успішної валідації файлу відбувається валідація обличчя на фото, а саме:

- перевірка чи на фото тільки одне обличчя;
- перевірка чи відкриті на обличчі очі.

Якщо всі етапи валідації пройдено успішно, то отримане фото зберігається в сховище даних Amazon S3. Після того, як фото буде збережено у системі зберігається запис дії про автентифікацію в системі для адміністратора.

Далі відбувається запит до Amazon Rekognition з іформацією про профіль та фото, яке надійшло для автентифікації. Сервіс обробляє отримані дані та повертає результат автентифікації, який далі передається у відповідь організації користувачу.

Відповідно до описаної вище послідовності дій, побудовано UML-діаграму послідовності, яка знаходиться у графічному матеріалі, лист 5.

4.3.3 Діаграма компонентів

Умовно Систему FaceAPI можна поділити на три компоненти:

- FaceAPI Client – клієнтська частина;
- FaceAPI Server – серверна частина;
- Amazon Services – сервіси Amazon.

Розглянемо детальніше кожен з цих компонентів та набори пакетів, які вони в собі містять.

FaceAPI Client

Цей компонент відповідає за клієнтську частину системи. Він містить увесь потрібний функціонал для адміністратора, а також демонстрацію роботи запитів на реєстрацію та авторизацію профілів до серверної частини. Даний компонент містить в собі наступний набір пакетів:

- handlers – містить функції, що обробляють запити, які надходять до компоненту;
- views – містить функції для відображення веб-сторінок;
- templates – містить шаблони веб-сторінок;
- services – містить функціонал для роботи з серверною частиною;
- utils – містить набір допоміжних функцій.

FaceAPI Server

Компонент FaceAPI Server відповідає за серверну частину системи. У ньому реалізовано функціонал для обробки запитів від клієнтської частини та REST API для того, щоб організації-користувачі могли користуватись системою. Даний компонент містить в собі наступний набір пакетів:

- handlers – містить функції, що обробляють запити, які надходять до компоненту;
- forms – містить функції для валідації даних;
- models – містить класи, які відповідають моделям бази даних;
- services – містить функціонал для роботи з сервісами Amazon;

– bl – містить функціонал, що реалізує бізнес-логіку системи FaceAPI;

– utils – містить набір допоміжних функцій.

Також даний компонент має зовнішній зв’язок з базою даних, яка містить дані про всі зареєстровані організації, їх користувачів та про дії в системі.

Amazon Services

Цей компонент відповідає за сервіси Amazon: S3 та Rekognition. Саме в S3 зберігаються всі файли зображень, які проходять через Систему FaceAPI, а сервіс Amazon Rekognition відповідає за розпізнавання та автентифікацію обличь.

У відповідності до описаних вище компонентів, побудовано UML-діаграму компонентів, яка знаходиться у графічному матеріалі, лист 6.

4.3.4 Специфікація функцій

Основним функціоналом даного дипломного проекту є REST API, яким будуть користуватись організації. Проведемо огляд доступних методів (далі – endpoint) цього API, функцій, які за них відповідають, та структуру HTTP-запиту до цього endpoint’у. Назву endpoint’у будемо представляти у форматі:

Endpoint <тип HTTP-запиту> <шлях до endpoint’у>

Endpoint GET /api/profiles

Оброблюється фукцією *profile_list(request)*, яка приймає на вхід запит та повертає у відповідь список усіх зареєстрованих профілів клієнтів для організації. Нижче наведено структуру HTTP-запиту:

```
GET /api/profiles HTTP/1.1
Host: <FaceAPI Server Host>
Content-Type: application/json
company_id: <ідентифікатор організації в Системі FaceAPI>
token: <ключ доступу організації>
```

Endpoint GET /api/profiles/{profile_id}

Оброблюється фукцією *get_profile(request)*, яка приймає на вхід запит та повертає у відповідь дані профілю клієнта, ідентифікатор якого відповідає значенню *profile_id* у запиті. У випадку, якщо профілю з таким ідентифікатором не існує, то відповідь серверу міститиме відповідне повідомлення. Нижче наведено структуру HTTP-запиту:

```
GET /api/profiles/<ідентифікатор профілю клієнта> HTTP/1.1
Host: <FaceAPI Server Host>
Content-Type: application/json
company_id: <ідентифікатор організації в Системі FaceAPI>
token: <ключ доступу організації>
```

Endpoint PATCH /api/profiles/{profile_id}

Оброблюється фукцією *patch_profile(request)*, яка приймає на вхід запит з зображенням та повертає у відповідь дані про профіль клієнта, ідентифікатор якого відповідає значенню *profile_id* у запиті, та інформацію про кількість фото, які необхідно завантажити для успішної реєстрації. Нижче наведено структуру HTTP-запиту:

```
PATCH /api/profiles/<ідентифікатор профілю клієнта>HTTP/1.1
Host: <FaceAPI Server Host>
Content-Type: multipart/form-data
company_id: <ідентифікатор організації в Системі FaceAPI>
token: <ключ доступу організації>
<байти зображення>
```

Endpoint POST /api/profiles/{profile_id}/auth

Оброблюється фукцією *auth_profile(request)*, яка приймає на вхід запит з зображенням та повертає у відповідь повідомлення про успішну або неуспішну автентифікацію обличчя для профілю, ідентифікатор якого відповідає значенню *profile_id* у запиті. Нижче наведено структуру HTTP-запиту:

```
POST /api/profiles/<ідентифікатор профілю клієнта>/auth HTTP/1.1
```

```
Host: <FaceAPI Server Host>
Content-Type: multipart/form-data
company_id: <ідентифікатор організації в Системі FaceAPI>
token: <ключ доступу організації>
<байти зображення>
```

Endpoint DELETE /api/profiles/{profile_id}

Оброблюється фукцією *delete_profile(request)*, яка приймає на вхід запит та повертає у відповідь повідомлення про видалення профілю, ідентифікатор якого відповідає значенню *profile_id* у запиті. Нижче наведено структуру HTTP-запиту:

```
DELETE /api/profiles/<ідентифікатор профілю клієнта> HTTP/1.1
Host: <FaceAPI Server Host>
Content-Type: application/json
company_id: <ідентифікатор організації в Системі FaceAPI>
token: <ключ доступу організації>
```

Можливі варіанти відповідей та структура відповідей описані в керівництві користувача (пункт 5.1.2).

Висновок до розділу

У цьому розділі було описано технології та засоби розробки програмного продукту для дипломного проекту, обґрунтовано їх вибір та розглянуто переваги та недоліки.

Описано архітектуру Системи FaceAPI та кожну архітектурну одиницю окремо. Наведено опис класів, компонентів системи та відображено на UML-діаграмах. Також побудовано UML-діаграму послідовності для процесу автентифікації обличчя клієнта.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Система FaceAPI має дві групи користувачів: адміністратор, який керує даними в системі й займається веденням організацій, та безпосередньо зареєстровані організації користувачі. Адміністратор взаємодіє з веб-інтерфейсом системи, а організація-користувач взаємодіє з REST API. Нажче наведено керівництво користувача для кожної групи.

5.1.1 Керівництво адміністратора

Для початку роботи адміністратор повинен авторизуватись в системі. На рисунку 5.1 зображено форму входу в Систему FaceAPI.

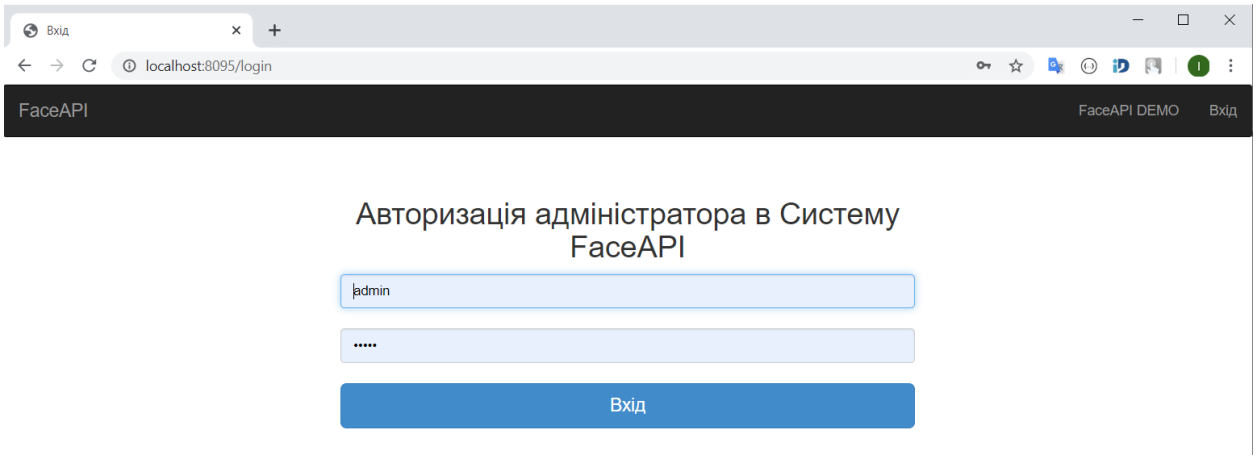


Рисунок 1.1 – Сторінка входу в систему

Після цього адміністратор отримує доступ до ведення організацій та до історії дій в системі. Безпосередньо після авторизації адміністратор потрапляє на сторінку з списком усіх зареєстрованих організацій, що зображена на рисунку 5.2.

Рисунок 1.2 – Сторінка з списком зареєстрованих організацій

Для того щоб згенерувати новий ключ доступу для конкретної організації потрібно натиснути на кнопку «Згенерувати новий» в рядку відповідної організації в колонці «Ключ доступу». На рисунку 5.3 показано результат генерації нового ключа доступу для організації з ідентифікатором «5ed2effc8c8b9ae67b4cf7c0».

Рисунок 1.3 – Новий ключ доступу для організації

Натиснувши на кнопку «Редагувати» у рядку потрібної організації, адміністратор перейде на сторінку редагування організації, що зображена на рисунку 5.4. На цій сторінці адміністратор має можливість змінити дані організації.

Рисунок 1.4 – Сторінка редагування організації

Результат редагування зображено на рисунку 5.5.

Рисунок 1.5 – Результат редагування організації

Для того щоб видалити організацію потрібно натиснути на кнопку «Видалити» у колонці «Керування» у рядку відповідної організації. На рисунку 5.6. показано результат видалення тестової організації з ідентифікатором «5ed2effc8c8b9ae67b4cf7c0».

Рисунок 1.6 – Результат видалення організації

Для того, щоб додати нову організацію адміністратору потрібно натиснути на кнопку «Додати нову організацію» після чого відкриється сторінка, яка зображена на рисунку 5.7.

Рисунок 1.7 – Сторінка створення нової організації

На цій сторінці потрібно ввести дані про організацію. У разі неправильного заповнення даних, після натиснення на кнопку «Зберегти» буде показано помилки введення даних, що зображені на рисунку 5.8.

Рисунок 1.8 – Результат валідації даних про організацію

На рисунку 5.9 зображено результат додавання нової організації.

5ed3adb1741866b323d2c38c	ТОВ ГеймСпейс	gamespace@gmail.com	Мережа комп'ютерних клубів	zgxyktBEK3aUNp8l	Згенерувати новий	Видалити
5ed3c60a741866b323d2c395	Нова тестова організація	new_test@gmail.com	Це тестова організація	j0SMLTWMmo5QWSZq	Згенерувати новий	Редагувати / Видалити

Рисунок 1.9 – Результат додавання нової організації

Адміністратор також має можливість переглянути історію дій в системі. Для цього потрібно натиснути кнопку «Історія дій» в меню. Сторінка з історією дій в системі зображена на рисунку 5.10.

Дата і час	Дія	ID Організації	ID Профілю
2020-05-31T15:42:13.458000+00:00	автентифікація профілю	5ed3c60a741866b323d2c395	2
2020-05-31T15:41:32.538000+00:00	додано фото до профілю	5ed3c60a741866b323d2c395	2
2020-05-31T15:41:11.629000+00:00	додано фото до профілю	5ed3c60a741866b323d2c395	2
2020-05-31T15:40:54.436000+00:00	створено новий профіль	5ed3c60a741866b323d2c395	2
2020-05-31T14:58:18.422000+00:00	створено компанію 5ed3c60a741866b323d2c395	admin	None
2020-05-31T14:38:41.933000+00:00	видалено компанію 5ed2effc8c8b9ae67b4cf7c0	admin	None
2020-05-31T14:36:13.244000+00:00	змінено дані компанії 5ed3ac5c741866b323d2c384	admin	None
2020-05-31T14:19:59.139000+00:00	згенеровано новий ключ доступу для компанії 5ed2effc8c8b9ae67b4cf7c0	admin	None
2020-05-31T13:16:05.112000+00:00	згенеровано новий ключ доступу для компанії 5ed3adb1741866b323d2c38c	admin	None
2020-05-31T13:16:03.472000+00:00	згенеровано новий ключ доступу для компанії 5ed3acf9741866b323d2c388	admin	None
2020-05-31T13:16:02.239000+00:00	згенеровано новий ключ доступу для компанії 5ed2effc8c8b9ae67b4cf7c0	admin	None
2020-05-31T13:16:00.973000+00:00	згенеровано новий ключ доступу для компанії 5ed3acab741866b323d2c386	admin	None
2020-05-31T13:14:25.669000+00:00	створено компанію 5ed3adb1741866b323d2c38c	admin	None
2020-05-31T13:13:30.942000+00:00	створено компанію 5ed3ad7a741866b323d2c38a	admin	None

Рисунок 1.10 – Сторінка історії дій в системі

5.1.1 Керівництво організації-користувача

Взаємодія організації користувача з Системою FaceAPI відбується через REST API та HTTP-запити. Користувач має можливість:

– додавати, редагувати та видаляти профілі своїх клієнтів;

– переглядати список доданих профілів клієнтів;

– виконувати автентифікацію обличчя клієнта.

Кожен HTTP-запит повинен містити два обов'язкових заголовки: *company_id* та *token*, які містять значення ідентифікатору організації та ключа доступу. Ці дані організація-користувач отримує після реєстрації. У відповіді на кожен запис повертається код відповіді та дані у форматі JSON.

До запитів на реєстрацію, редагування та автентифікацію профілю клієнта потрібно додавати фото клієнта. Фото повинне відповідати наступним критеріям:

– обличчя на фото повинно займати більшість площі фото;

– очі на фото повинні бути відкритими;

– обличчя на фото має бути в анфас, повернене вправо або вліво.

Розглянемо структуру та відповіді кожного HTTP-запиту. Для демонстрації результатів запитів в якості *company_id* та *token* взято *5ed3ac5c741866b323d2c384* та *QpmWZhRCSLwaQJxg* відповідно.

Запит на додавання та редагування профілю клієнта

Запит має наступну структуру:

```
POST /api/profiles/<ідентифікатор профілю клієнта>/auth HTTP/1.1
Host: <FaceAPI Server Host>
Content-Type: multipart/form-data
company_id: <ідентифікатор організації в Системі FaceAPI>
token: <ключ доступу організації>
<байти зображення>
```

На рисунку 5.11 зображено відповідь на запит.

```
{
    "data": {
        "active": false,
        "photos": [
            {
                "face_id": "1a14b134-42fa-47b5-94d3-c69351a00413",
                "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_0"
            }
        ],
        "company_id": "5ed3ac5c741866b323d2c384",
        "profile_id": "1",
        "id": "5ed3d8ab741866b323d2c39c",
        "main_photo": {
            "face_id": "1a14b134-42fa-47b5-94d3-c69351a00413",
            "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_0"
        }
    },
    "registration_complete": false,
    "photo_remain": 2
}
```


Рисунок 1.11 – Відповідь на запит реєстрації профілю клієнта

Відповідь на даний запит містить інформацію про ідентифікатори компанії, клієнту та профілю, інформацію про завантажені фото, а також про статус реєстрації та кількість фото, що залишилась для успішної реєстрації. Для повної реєстрації профілю потрібно відправити 3 запити з трьома фото клієнта. На рисунку 5.12 показано відповідь на третій запит з третім фото.

```
{
    "data": {
        "active": true,
        "photos": [
            {
                "face_id": "1a14b134-42fa-47b5-94d3-c69351a00413",
                "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_0"
            },
            {
                "face_id": "8937924c-94ab-4e16-8f5e-4a9352da73e5",
                "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_1"
            },
            {
                "face_id": "3e07b4ed-2e4b-47e1-8b3d-7e13386b8741",
                "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_2"
            }
        ],
        "company_id": "5ed3ac5c741866b323d2c384",
        "profile_id": "1",
        "id": "5ed3d8ab741866b323d2c39c",
        "main_photo": {
            "face_id": "1a14b134-42fa-47b5-94d3-c69351a00413",
            "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_0"
        }
    },
    "registration_complete": true,
    "photo_remain": 0
}
```

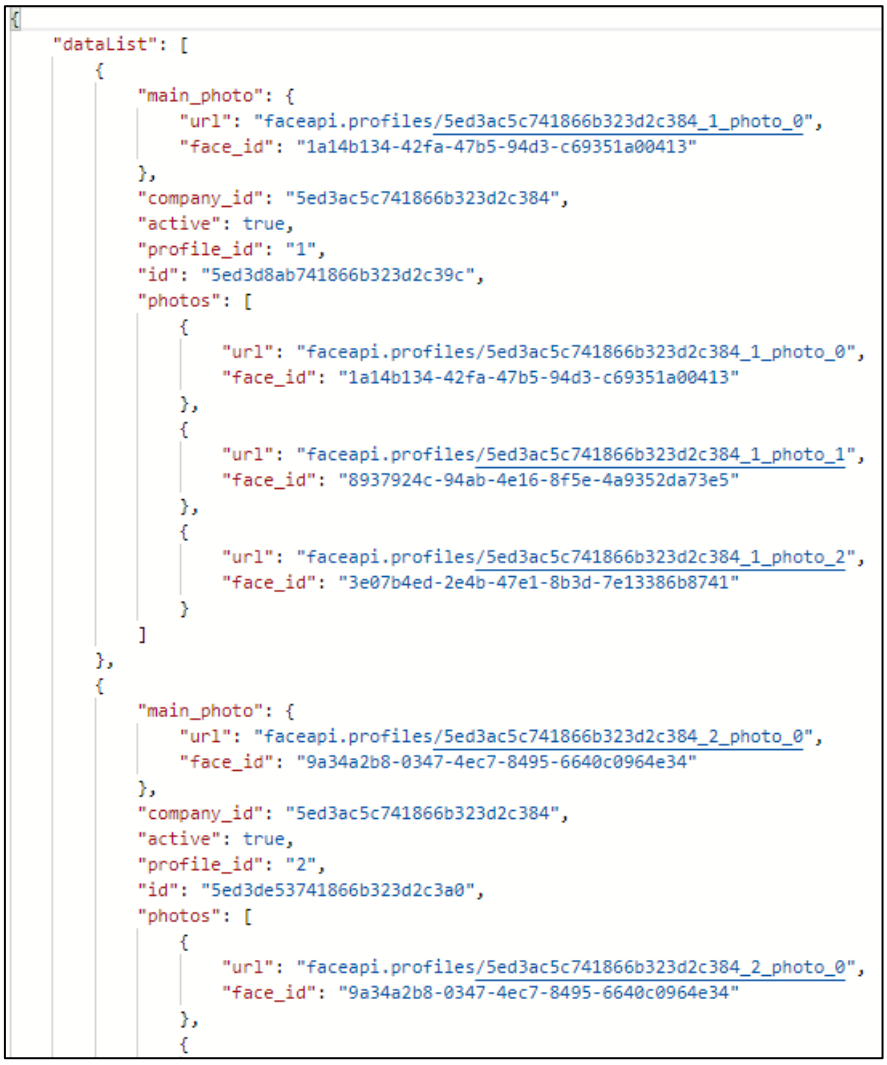

Рисунок 1.12 – Відповідь при успішній реєстрації профілю клієнта

Запит на отримання списку всіх доданих профілів клієнтів

Запит має наступну структуру:

```
GET /api/profiles HTTP/1.1

Host: <FaceAPI Server Host>

Content-Type: application/json

company_id: <ідентифікатор організації в Системі FaceAPI>

token: <ключ доступу організації>
```

Відповіддю на даний запит є список усіх зареєстрованих даною організацією профілів клієнтів. Приклад відповіді зображено на рисунку 5.13.



```
{
    "dataList": [
        {
            "main_photo": {
                "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_0",
                "face_id": "1a14b134-42fa-47b5-94d3-c69351a00413"
            },
            "company_id": "5ed3ac5c741866b323d2c384",
            "active": true,
            "profile_id": "1",
            "id": "5ed3d8ab741866b323d2c39c",
            "photos": [
                {
                    "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_0",
                    "face_id": "1a14b134-42fa-47b5-94d3-c69351a00413"
                },
                {
                    "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_1",
                    "face_id": "8937924c-94ab-4e16-8f5e-4a9352da73e5"
                },
                {
                    "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_1_photo_2",
                    "face_id": "3e07b4ed-2e4b-47e1-8b3d-7e13386b8741"
                }
            ]
        },
        {
            "main_photo": {
                "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_2_photo_0",
                "face_id": "9a34a2b8-0347-4ec7-8495-6640c0964e34"
            },
            "company_id": "5ed3ac5c741866b323d2c384",
            "active": true,
            "profile_id": "2",
            "id": "5ed3de53741866b323d2c3a0",
            "photos": [
                {
                    "url": "faceapi.profiles/5ed3ac5c741866b323d2c384_2_photo_0",
                    "face_id": "9a34a2b8-0347-4ec7-8495-6640c0964e34"
                },
                {
```


Рисунок 1.13 – Відповідь з списком даних про зареєстрованих клієнтів

Структуру даних кожного профіля наведено в описі запиту вище.

Запит на автентифікацію профілю

Запит має наступну структуру:

```
POST /api/profiles/<ідентифікатор профілю клієнта>/auth HTTP/1.1
Host: <FaceAPI Server Host>
Content-Type: multipart/form-data
company_id: <ідентифікатор організації в Системі FaceAPI>
token: <ключ доступу організації>
<байти зображення>
```

На рисунках 5.14 та 5.15 наведено варіанти відповіді успішної та неуспішної автентифікації клієнта відповідно.

Рисунок 1.14 – Відповідь про успішну автентифікацію

Рисунок 1.15 – Відповідь про неуспішну автентифікацію

Запит на видалення профілю

Запит має наступну структуру:

```
DELETE /api/profiles/<ідентифікатор профілю клієнта> HTTP/1.1
Host: <FaceAPI Server Host>
Content-Type: application/json
company_id: <ідентифікатор організації в Системі FaceAPI>
token: <ключ доступу організації>
```

Після видалення профілю надсилається відповідь, яка зображена на рисунку 5.16.

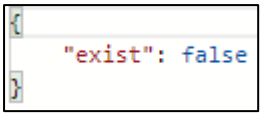


Рисунок 1.16 – Відповідь при видаленні профілю клієнта

5.2 Випробування програмного продукту

5.2.1 Мета випробувань

Метою випробувань розроблюваного програмного продукту являється перевірка функцій Системи FaceAPI на відповідність вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проведено у відповідності до наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

Функціональність Системи FaceAPI можна розділити на функціональність для адміністратора та для організації-користувача. Для кожного типу функціональності було створено відповідніс тести та перевірено програмний продукт на відповідність наведеним у технічному завданні вимогам. Під час тестування було перевірено функціональність адміністратора та організації-користувача.

У таблицях 5.1-5.6 наведено перелік випробувань функціоналу для адміністратора.

Таблиця 1.1 – Тестування введення логіну та паролю

Мета тесту	**Перевірка функції «Вхід адміністратора»**
Початковий стан Системи FaceAPI	Відкрита сторінка «Вхід»
Вхідні дані	Логін та пароль адміністратора
Схема проведення тесту	Ввести логін та пароль адміністратора у відповідні поля
Очікуваний результат	Вхід в систему виконано, відкрито сторінку «Список організацій»
Стан Системи FaceAPI після проведення випробування	Вхід в систему виконано, відкрито сторінку «Список організацій»

Таблиця 1.2 – Тестування додавання нової організації

Мета тесту	**Перевірка функції «Додавання організації»**
Початковий стан Системи FaceAPI	Відкрита сторінка «Нова організація»
Вхідні дані	Назва, email та додаткова інформація організації
Схема проведення тесту	Ввести відповідні дані про організацію у відповідні поля на сторінці
Очікуваний результат	Додано нову організацію, згенеровано унікальний ідентифікатор та ключ доступу, відкрито сторінку «Список організацій», де відображається щойно додана організація
Стан Системи FaceAPI після проведення випробування	Додано нову організацію, згенеровано унікальний ідентифікатор та ключ доступу, відкрито сторінку «Список організацій», де відображається щойно додана організація

Таблиця 1.3 – Тестування редагування організації

Мета тесту	**Перевірка функції «Редагування організації»**
Початковий стан Системи FaceAPI	Відкрита сторінка «Редагування організації», яка містить дані про обрану організацію у відповідних полях
Вхідні дані	Нова назва організації
Схема проведення тесту	Змінити назву організації у відповідному полі
Очікуваний результат	Змінено дані про організацію, відкрито сторінку «Список організацій», де відображаються зміни
Стан Системи FaceAPI після проведення випробування	Змінено дані про організацію, відкрито сторінку «Список організацій», де відображаються зміни

Таблиця 1.4 – Тестування генерації нового ключа доступу

Мета тесту	**Перевірка функції «Редагування організації»**
Початковий стан Системи FaceAPI	Відкрита сторінка «Список організацій»
Вхідні дані	Відсутні
Схема проведення тесту	Натиснути кнопку «Згенерувати новий» у колонці «Ключ доступу» у відповідному рядку організації
Очікуваний результат	Змінено ключ доступу організації, відкрито сторінку «Список організацій», де відображаються зміни
Стан Системи FaceAPI після проведення випробування	Змінено ключ доступу організації, відкрито сторінку «Список організацій», де відображаються зміни

Таблиця 1.5 – Тестування видалення організації

Мета тесту	**Перевірка функції «Видалення організації»**
Початковий стан Системи FaceAPI	Відкрита сторінка «Список організацій»
Вхідні дані	Відсутні
Схема проведення тесту	Натиснути кнопку «Видалити» у колонці «Керування» у відповідному рядку організації
Очікуваний результат	Видалено дані про організацію та зареєстровані профілі клієнтів, відкрито сторінку «Список організацій», де відсутні дані про видалену організацію
Стан Системи FaceAPI після проведення випробування	Видалено дані про організацію та зареєстровані профілі клієнтів, відкрито сторінку «Список організацій», де відсутні дані про видалену організацію

Таблиця 1.6 – Тестування перегляду історії дій

Мета тесту	**Перевірка функції «Видалення організації»**
Початковий стан Системи FaceAPI	Відкрита сторінка «Список організацій»
Вхідні дані	Відсутні
Схема проведення тесту	Вибрати пункт меню «Історія дій»
Очікуваний результат	Відкрито сторінку «Історія дій», де відображається список дій
Стан Системи FaceAPI після проведення випробування	Відкрито сторінку «Історія дій», де відображається список дій

У таблицях 5.7-5.9 наведено перелік випробувань функціоналу для організації-користувача.

Редагування та додавання нових профілів клієнтів виконуються за допомогою одного HTTP-запиту, який полягає в завантаженні фото до профілю. Тому функціонал додавання та редагування профілю буде перевірено в одному тесті.

Таблиця 1.7 – Тестування додавання та редагування профілю клієнта

Мета тесту	**Перевірка функцій «Додавання профілю клієнта» та «Редагування профілю клієнта»**
Початковий стан Системи FaceAPI	Очікування HTTP-запитів
Вхідні дані	Ідентифікатор організації, ключ доступу організації, ідентифікатор клієнта, фото клієнта
Схема проведення тесту	Вказати в заголовках запиту ідентифікатор організації та ключ доступу, в рядку запиту вказати ідентифікатор клієнта, в тіло запиту додати фото, відправити запит
Очікуваний результат	Відповідь з інформацією про профіль, стан реєстрації та кількість фото для завершення реєстрації
Відповідь від Системи FaceAPI після проведення випробування	Відповідь з інформацією про профіль, стан реєстрації та кількість фото для завершення реєстрації

Таблиця 1.8 – Тестування видалення профілю клієнта

Мета тесту	**Перевірка функції «Видалення профілю клієнта»**
Початковий стан Системи FaceAPI	Очікування HTTP-запитів
Вхідні дані	Ідентифікатор організації, ключ доступу організації, ідентифікатор клієнта
Схема проведення тесту	Вказати в заголовках запиту ідентифікатор організації та ключ доступу, в рядку запиту вказати ідентифікатор клієнта, відправити запит
Очікуваний результат	Видалено дані профілю, відповідь з повідомленням, що профіль видалено
Відповідь від Системи FaceAPI після проведення випробування	Видалено дані профілю, відповідь з повідомленням, що профіль видалено

Таблиця 1.9 – Тестування автентифікації клієнта

Мета тесту	**Перевірка функції «Автентифікація клієнта»**
Початковий стан Системи FaceAPI	Очікування HTTP-запитів
Вхідні дані	Ідентифікатор організації, ключ доступу організації, ідентифікатор клієнта, фото клієнта
Схема проведення тесту	Вказати в заголовках запиту ідентифікатор організації та ключ доступу, в рядку запиту вказати ідентифікатор клієнта, в тіло запиту додати фото клієнта, відправити запит

Продовження таблиці 5.9

Мета тесту	**Перевірка функції «Автентифікація клієнта»**
Очікуваний результат	Відповідь з повідомленням про успішну автентифікацію
Відповідь від Системи FaceAPI після проведення випробування	Відповідь з повідомленням про успішну автентифікацію

Висновок до розділу

У цьому розділі надано інструкція користувача для адміністратора та організації. В інструкції описані дії, які мають можливість виконувати користувачі.

Також були проведені функціональні випробування програмного продукту, які підтвердили його працездатність.

ЗАГАЛЬНІ ВИСНОВКИ

У даному дипломному проекті було розглянуто питання спрощення процесу підтвердження платежу, для вирішення якого було розроблено Систему FaceAPI, яка дозволяє організаціям отримати доступ до функціоналу автентифікації обличчя по фото й вбудувати в свою систему функціонал підтвердження платежів за допомогою обличчя.

У розділі загальних положень було проаналізовано предметну область і наведено опис процесу використання розробленого програмного продукту. Також описано процес діяльності, виділено групи користувачів та визначено функції, які вони можуть виконувати.

У інформаційному розділі проведено огляд вхідних та вихідних даних, на основі яких було обрано сховища даних та сформовано їх структуру.

У розділі математичного забезпечення було проведено опис процесу розпізнавання обличчя та засобів для реалізації даної задачі.

В розділі програмного та технічного забезпечення були описані обрані для розробки технології та засоби й наведено обґрунтування вибору саме їх. Також наведено опис загальної архітектури Системи FaceAPI, конкретних класів та функцій. Побудовано UML-діаграми класів, компонентів та UML-діаграму послідовності для процесу автентифікації обличчя.

Наведено інстукцію використання для адміністратора та для організації-користувача, а також проведено випробування програмного продукту, по результатах якого було підтверджено його працездатність.

ПЕРЕЛІК ПОСИЛАНЬ

1. Nikhil B. Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms / Buduma Nikhil., 2017. – 277 с.
2. Лутц М. Изучаем Python / Марк Лутц. – Санкт-Петербург: Диалектика, 2019. – 832 с.
3. M. a. Turk and A. P. Pentland, “Face Recognition Using Eigenfaces,” Journal of Cognitive Neuroscience, vol. 3, no. 1. pp. 72–86, 1991.
4. J. Wright, a. Y. Yang, a. Ganesh, S. S. Sastry, and Y. Ma, “Robust face recognition via sparse representation.,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 31, no. 2, pp. 210–227, 2009.
5. P. Sukhija, S. Behal, and P. Singh, “Face Recognition System Using Genetic Algorithm,” in Procedia Computer Science, 2016, vol. 85.
6. Aviani G. HTTP and everything you need to know about it [Електронний ресурс] / Goran Aviani. – 2018. – Режим доступу до ресурсу: https://medium.com/faun/http-and-everything-you-need-to-know-about-it-8273bc224491.
7. Портяний І.С. Використання згорткових нейронних мереж у задачі розпізнавання обличчя // Матеріали ІІІ всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2020) – м. Київ.: НТУУ «КПІ ім. Ігоря Сікорського», 24-30 квітня 2020 р. – С. 130-135
8. AliPay [Електронний ресурс] / Alipay – Режим доступу до ресурсу: https://intl.alipay.com/.
9. FacePay24 [Електронний ресурс] – Режим доступу до ресурсу: https://privatbank.ua/.
10. Brownlee J. A Gentle Introduction to Deep Learning for Face Recognition [Електронний ресурс] / Jason Brownlee. – 2019. – Режим доступу до ресурсу: https://machinelearningmastery.com/introduction-to-deep-learning-for-face-recognition/.

11. R. Xia, J. Deng, B. Schuller, and Y. Liu, “Modeling gender information for emotion recognition using Denoising autoencoder,” in ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, 2014, pp. 990–994.
12. G. E. Hinton, S. Osindero, and Y. W. Teh, “A fast learning algorithm for deep belief nets.,” Neural Comput., vol. 18, no. 7, pp. 1527–54, 2006.[38] Y. Bengio, Learning Deep Architectures for AI, vol. 2, no. 1. 2009.
13. Rohan T. Convolutional Networks for everyone [Електронний ресурс] / Rohan Thomas. – 2018. – Режим доступу до ресурсу: https://medium.com/@rohanthomas.me/convolutional-networks-for-everyone-1d0699de1a9d.
14. Raimi K. Illustrated: 10 CNN Architectures [Електронний ресурс] / Karim Raimi. – 2019. – Режим доступу до ресурсу: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d.
15. Amazon Rekognition [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/rekognition/.
16. An Introduction to Asynchronous Programming in Python [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://medium.com/velotio-perspectives/an-introduction-to-asynchronous-programming-in-python-af0189a88bbb.
17. Amazon S3 [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/s3/.
18. Редмонд Э. Семь баз данных за семь недель / Э. Редмонд, Д. Р. Уилсон. – Москва: ДМК Пресс, 2013. – 384 с.
19. Змерзлий І. Клієнт-серверна архітектура та ролі серверів [Електронний ресурс] / Іван Змерзлий. – 2017. – Режим доступу до ресурсу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229.

20. Ivashchenko A. REST: простым языком [Електронний ресурс] / Andriy Ivashchenko. – 2019. – Режим доступу до ресурсу: https://medium.com/@andr.ivas12/rest-простым-языком-90a0bca0bc78.

Додаток А

Тексти програмного коду

Підсистема розпізнавання та автентифікації обличь в системі оплати

(Найменування програми (документа))

DVD-R

(Вид носія даних)

36 арк, 274 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2020 року

faceapi/app/__main__.py

```
import logging

import aiohttp_autoreload
from aiohttp import web

from .application import create_app

log = logging.getLogger(__name__)


if __name__ == '__main__':
    # Create app
    app = create_app()

    app_config = app['config']['app']

    if app_config['debug']:
        aiohttp_autoreload.start()

    log.info('Listen on {}:{}'.format(app_config['host'], app_config['port']))

    web.run_app(app, port=app_config['port'])
```

faceapi/app/application.py

```
import logging
from aiohttp import web

from app import signals
from app.config import get_config
from app.routes import setup_routes

log = logging.getLogger(__name__)


def create_app() -> web.Application:
    """create application"""
    config = get_config()

    # App configuration
    app_config = config['app']

    # Create app
    app = web.Application(
        debug=app_config['debug']
    )
    app['config'] = config

    # Setup signals
    # To run on app startup
    app.on_startup.extend(
        [
            signals.mongodb_connect,
            signals.init_s3,
            signals.init_rekognition,
        ]
    )

    # To run before shutdown
    app.on_cleanup.extend(
        [
            signals.disconnect_mongodb,
        ]
    )

    # Routes
```

```
    setup_routes(app)

    # Logging
    logging.basicConfig(level=logging.INFO)

    return app
```

faceapi/app/application.py

```
import logging
import aiohttp_cors

from aiohttp import web

from app import signals
from app.config import get_config
# from app.const import APP_DIR
from app.routes import setup_routes

log = logging.getLogger(__name__)


def create_app() -> web.Application:
    """create application"""
    config = get_config()

    # App configuration
    app_config = config['app']

    # Create app
    app = web.Application(
        debug=app_config['debug']
    )
    app['config'] = config

    # Setup signals
    # To run on app startup
    app.on_startup.extend(
        [
            signals.mongodb_connect,
            signals.init_s3,
            signals.init_rekognition,
        ]
    )

    # To run before shutdown
    app.on_cleanup.extend(
        [
            signals.disconnect_mongodb,
        ]
    )

    # Routes
    setup_routes(app)

    # Logging
    logging.basicConfig(level=logging.INFO)

    return app
```

faceapi/app/security.py

```
import inspect
import logging
from typing import Any, Callable, Dict, Optional

from aiohttp import web
```

```python
from app.const import Permission
from app.models.company import Company

log = logging.getLogger(__name__)


async def check_permission(request, permission, context=None):
    allowed = await permits(request, permission, context=context)
    if not allowed:
        raise web.HTTPForbidden()


def has_permission(
    permission: Permission, context_factory: Optional[Any] = None,
) -> Callable[..., Any]:
    """ context factory must be None or callable / coroutine object
    """
    if not any(
        [
            context_factory is None,
            inspect.iscoroutinefunction(context_factory),
            callable(context_factory),
        ]
    ):
        msg = (
            "Wrong context_factory type. "
            "Expected None, coroutine or normal callable"
        )
        raise RuntimeError(msg)

    def wrapper(fn: Callable[..., Any]) -> Callable[..., Any]:
        async def wrapped(*args: Any, **kwargs: Dict[Any, Any]) -> Any:
            request = args[-1]
            if not isinstance(request, web.BaseRequest):
                raise RuntimeError(
                    "Incorrect decorator usage. "
                    "Expecting `def handler(request)` "
                    "or `def handler(self, request)`."
                )

            context: Any = None
            if not context_factory:
                raise RuntimeError(
                    "Wrong context_factory type. "
                    "Expected None, coroutine or normal callable"
                )
            if inspect.iscoroutinefunction(context_factory):
                context = await context_factory(request)
            elif callable(context_factory):
                context = context_factory(request)

            if context_factory:
                kwargs['context'] = context

            allowed = await permits(request, permission, context)

            if not allowed:
                raise web.HTTPForbidden
            ret = await fn(*args, **kwargs)
            return ret

        return wrapped

    return wrapper
async def permits(
    request: web.BaseRequest, permission: Permission, context: Any = None
) -> bool:
```

```
    if permission == Permission.company_edit:
        return await _permits_company_edit(request, context)
    if permission == Permission.profile_edit:
        return await _permits_profile_edit(request, context)
    else:
        raise ValueError(f'No algo for {permission} defined')


async def _permits_company_edit(
    request: web.BaseRequest,
    context: Company
) -> bool:
    return True


async def _permits_profile_edit(
    request: web.BaseRequest,
    context
) -> bool:
    company = context.company
    token = context.token

    return all([
        company.token == token,
    ])
```

faceapi/app/signals.py

```
import asyncio
import boto3
import logging

from aiohttp import web
from motor.motor_asyncio import AsyncIOMotorClient

from app.models import instance

log = logging.getLogger(__name__)

async def mongodb_connect(app: web.Application) -> None:
    mongodb_config = app['config']['mongo']
    mongo_url = mongodb_config['uri']
    mongo_db = mongodb_config['db']
    # mongo_collection = mongodb_config['collection']

    log.info(f'mongo_url: {mongo_url}')
    mongo_client = AsyncIOMotorClient(
        mongo_url, io_loop=app.get('loop', asyncio.get_event_loop())
    )
    log.info(await mongo_client.list_databases())
    database = mongo_client[mongo_db]
    app['mongo_client'] = mongo_client
    app['db'] = database
    instance.init(database)
    log.info('Connected to MongoDB')

async def disconnect_mongodb(app: web.Application) -> None:
    client = app['mongo_client']
    client.close()
    log.info('MongoDB disconnected')

async def init_s3(app: web.Application) -> None:
    aws_config = app['config']['aws']
    app['s3_client'] = boto3.client(
        's3',
        aws_access_key_id=aws_config.get('access_key', None),
        aws_secret_access_key=aws_config.get('secret_key', None),
        region_name=aws_config.get('region', None),
```

```
    )
    log.info('S3 initialized')

async def init_rekognition(app: web.Application) -> None:
    aws_config = app['config']['aws']
    app['rekognition_client'] = boto3.client(
        'rekognition',
        aws_access_key_id=aws_config.get('access_key', None),
        aws_secret_access_key=aws_config.get('secret_key', None),
        region_name=aws_config.get('region', None),
    )
    log.info('Rekognition initialized')
```

faceapi/app/models/__init__.py

```
from umongo import MotorAsyncIOInstance

instance = MotorAsyncIOInstance()
```

faceapi/app/models/action.py

```
from umongo import Document, fields

from . import instance


@instance.register
class Action(Document):
    class Meta:
        strict = False

    company_id = fields.StrField(required=True)
    profile_id = fields.StrField()
    action = fields.StrField(required=True)
    datetime = fields.DateTimeField(required=True)
```

faceapi/app/models/company.py

```
import enum

from umongo import Document, fields

from . import instance
from ..utils.common import generate_token


@enum.unique
class CompanyStatus(enum.Enum):
    active = 'active'
    deactivated = 'deactivated'


@instance.register
class Company(Document):
    class Meta:
        strict = False

    id = fields.ObjectIdField(attribute='_id')
    name = fields.StrField(required=True)
    email = fields.EmailField(required=True)
    token = fields.StrField(required=True)
    information = fields.StrField()
    status = fields.StrField(default=CompanyStatus.deactivated.value)

    def regenerate_token(self):
        self.token = generate_token()
        return self.token

    def is_activated(self):
```

```
        return self.status == CompanyStatus.active.value

    def activate(self):
        self.status = CompanyStatus.active.value

    def deactivate(self):
        self.status = CompanyStatus.deactivated.value
```

faceapi/app/models/photo.py

```
from umongo import EmbeddedDocument, fields

from . import instance


@instance.register
class Photo(EmbeddedDocument):
    class Meta:
        strict = False

    face_id = fields.StrField(required=True)
    url = fields.StrField(required=True)
```

faceapi/app/models/profile.py

```
from umongo import Document, fields

from . import instance
from .photo import Photo as mPhoto

@instance.register
class Profile(Document):
    class Meta:
        strict = False

    profile_id = fields.StrField(unique=True)
    company_id = fields.StrField()
    photos = fields.ListField(
        fields.EmbeddedField(mPhoto), default=[]
    )
    main_photo = fields.EmbeddedField(mPhoto)
    active = fields.BoolField(default=False)

    def activate(self):
        self.active = True

    def deactivate(self):
        self.active = False
```

faceapi/app/handlers/action.py

```
import logging
import umongo

from aiohttp import web

from app.const import Permission
from app.models.action import Action

from app.security import check_permission
from app.utils.exceptions import make_api_response

log = logging.getLogger(__name__)


async def action_list(
    request: web.Request
```

```
) -> web.Response:
    await check_permission(request, Permission.company_edit)

    actions_lst = Action.find({}).sort('datetime', -1)
    actions = [
        action.dump()
        async for action in actions_lst
    ]

    return make_api_response(
        web.HTTPOk,
        body={
            'dataList': actions,
        },
    )
```

faceapi/app/handlers/company.py

```
import logging
import typing as t

from aiohttp import web
from bson import ObjectId

from app.bl.action import log_action
from app.const import Permission
from app.forms import schema_load
from app.forms.company import CompanySchema
from app.models.company import Company, CompanyStatus
from app.security import has_permission, check_permission
from app.services import rekognition
from app.utils.common import collection_name
from app.utils.exceptions import make_api_response

log = logging.getLogger(__name__)


async def _context_factory(request: web.Request):
    try:
        company_id = ObjectId(request.match_info['company_id'])
        company = await Company.find_one(
            {'_id': company_id}
        )
    except Exception as e:
        raise make_api_response(
            web.HTTPNotFound,
            body={
                'error': {
                    'msg': str(e)
                },
                'data': {},
            },
        )
    if not company:
        raise make_api_response(
            web.HTTPNotFound,
            body={
                'error': {
                    'msg': f'Company _id {company_id} does not exists'
                },
                'data': {},
            },
        )
    return company


@has_permission(
    Permission.company_edit,
```

```
    context_factory=_context_factory
)
async def get_company(
    request: web.Request,
    context: Company
) -> web.Response:
    company = context

    return make_api_response(
        web.HTTPOk,
        body={
            'data': company.dump()
        },
    )


async def post_company(
    request: web.Request
) -> web.Response:
    await check_permission(request, Permission.company_edit)

    try:
        data = await request.json()
    except Exception as e:
        return make_api_response(web.HTTPUnprocessableEntity, text=str(e))

    schema = CompanySchema()
    result = schema_load(schema=schema, data=data)

    if result.errors:
        return make_api_response(
            web.HTTPUnprocessableEntity,
            body={'error': result.errors, 'body': {}}
        )
    company = Company(**result.data)

    company.activate()
    company.regenerate_token()

    await company.commit()

    await log_action(
        action=f'створено компанію {str(company.id)}'
    )

    collection_id = collection_name(company.id)
    await rekognition.create_collection(request.app, collection_id)

    return make_api_response(
        web.HTTPOk,
        body={
            'data': company.dump(),
        },
    )


@has_permission(
    Permission.company_edit,
    context_factory=_context_factory
)
async def patch_company(
    request: web.Request,
    context: Company
) -> web.Response:

    company = context
```

```
    try:
        data = await request.json()
    except Exception as e:
        return make_api_response(web.HTTPUnprocessableEntity, text=str(e))

    schema = CompanySchema(context=company)
    result = schema_load(schema=schema, data=data)

    if result.errors:
        return make_api_response(
            web.HTTPUnprocessableEntity,
            body={'error': result.errors, 'body': {}}
        )

    company.update(result.data)
    await company.commit()

    await log_action(
        action=f'змінено дані компанії {str(company.id)}'
    )

    return make_api_response(
        web.HTTPOk,
        body={
            'data': company.dump(),
        },
    )


@has_permission(
    Permission.company_edit,
    context_factory=_context_factory
)
async def delete_company(
    request: web.Request,
    context: Company
) -> web.Response:

    company = context

    company.deactivate()
    await company.commit()

    await log_action(
        action=f'видалено компанію {str(company.id)}'
    )

    # TODO: remove
    collection_id = collection_name(company.id)
    await rekognition.delete_collection(request.app, collection_id)

    return make_api_response(
        web.HTTPOk,
        body={
            'status': 'ok',
            'activated': company.is_activated()
        },
    )


@has_permission(
    Permission.company_edit,
    context_factory=_context_factory
)
async def regenerate_company_token(
    request: web.Request,
    context: Company
```

```
) -> web.Response:

    company = context

    company.regenerate_token()
    await company.commit()

    await log_action(
        action=f'згенеровано новий ключ доступу для компанії {str(company.id)}'
    )

    return make_api_response(
        web.HTTPOk,
        body={
            'status': 'ok',
            'token': company.token
        },
    )


async def company_list(
    request: web.Request
) -> web.Response:
    await check_permission(request, Permission.company_edit)

    companies_lst = Company.find(
            {'status': CompanyStatus.active.value}
    )
    companies = [
        company.dump()
        async for company in companies_lst
    ]

    return make_api_response(
        web.HTTPOk,
        body={
            'dataList': companies,
        },
    )
```

faceapi/app/handlers/profile.py

```
import logging
import typing as t

from aiohttp import web
from dataclasses import dataclass

from bson import ObjectId

from app.bl.profile import (
    create_new_profile,
    photo_count,
    delete_profile as _delete_profile,
    auth_profile as _auth_profile,
    add_photo_to_profile,
    save_auth_photo_to_s3
)
from app.const import Permission, REQUIRED_PHOTO_COUNT
from app.models.company import Company
from app.models.profile import Profile
from app.security import has_permission, check_permission
from app.utils.exceptions import make_api_response
from app.utils.validation import validate_photo

log = logging.getLogger(__name__)
```

```python
@dataclass(frozen=True)
class ProfileContext:
    profile_id: str
    profile: t.Optional[Profile]
    company_id: str
    token: str
    company: Company


async def _context_factory(request: web.Request):
    profile_id = request.match_info['profile_id']
    headers = request.headers
    company_id = headers.get('company_id')
    try:
        company = await Company.find_one(
            {'_id': ObjectId(company_id)}
        )
    except Exception as e:
        raise make_api_response(
            web.HTTPNotFound,
            body={
                'error': {
                    'msg': str(e)
                },
                'data': {},
            },
        )
    if not company:
        raise make_api_response(
            web.HTTPNotFound,
            body={
                'error': {
                    'msg': f'Company _id {company_id} does not exists'
                },
                'data': {},
            },
        )
    profile = await Profile.find_one(
        {
            'profile_id': profile_id,
            'company_id': company_id
        }
    )
    if not profile:
        return ProfileContext(
            profile_id=profile_id,
            profile=None,
            company_id=company_id,
            token=headers.get('token'),
            company=company
        )

    return ProfileContext(
        profile_id=profile_id,
        profile=profile,
        company_id=company_id,
        token=headers.get('token'),
        company=company
    )


@has_permission(
    Permission.profile_edit,
    context_factory=_context_factory
)
async def get_profile(
    request: web.Request,
```

```
    context: ProfileContext
) -> web.Response:
    profile = context.profile

    if not profile:
        raise make_api_response(
            web.HTTPNotFound,
            body={
                'error': {
                    'msg': f'Profile profile_id {context.profile} '
                           f'does not exists'
                },
                'data': {},
            },
        )

    log.info('Getting profile')

    return make_api_response(
        web.HTTPOk,
        body={
            'data': profile.dump()
        },
    )


@has_permission(
    Permission.profile_edit,
    context_factory=_context_factory
)
async def patch_profile(
    request: web.Request,
    context: ProfileContext
) -> web.Response:
    if request.headers.get('client'):
        try:
            photo_bytes = await request.read()
        except Exception as e:
            return make_api_response(web.HTTPUnprocessableEntity, text=str(e))
    else:
        try:
            reader = await request.multipart()
        except Exception as e:
            return make_api_response(web.HTTPUnprocessableEntity, text=str(e))

        image = await reader.next()
        photo_bytes = await image.read(decode=True)

    valid_photo, errors = await validate_photo(
        request.app,
        photo_bytes,
        registration=True
    )
    if not valid_photo:
        return make_api_response(
            web.HTTPUnprocessableEntity,
            body={
                'errors': errors,
            },
        )

    profile = context.profile
    if not profile:
        log.info('Creating new profile.')
        profile = await create_new_profile(request.app, context, photo_bytes)

        photo_num = photo_count(profile)
```

```
            return make_api_response(
                web.HTTPOk,
                body={
                    'data': profile.dump(),
                    'registration_complete': photo_num == REQUIRED_PHOTO_COUNT,
                    'photo_remain': REQUIRED_PHOTO_COUNT - photo_num
                },
            )

        photo_num = photo_count(profile)

        if photo_num == REQUIRED_PHOTO_COUNT:
            log.info('Deleting profile.')
            await _delete_profile(request.app, context)

            log.info('Creating new profile.')
            profile = await create_new_profile(request.app, context, photo_bytes)

            photo_num = photo_count(profile)
            return make_api_response(
                web.HTTPOk,
                body={
                    'data': profile.dump(),
                    'registration_complete': photo_num == REQUIRED_PHOTO_COUNT,
                    'photo_remain': REQUIRED_PHOTO_COUNT - photo_num
                },
            )

        log.info('Adding photo to profile.')
        profile = await add_photo_to_profile(request.app, context, photo_bytes)
        photo_num = photo_count(profile)
        return make_api_response(
            web.HTTPOk,
            body={
                'data': profile.dump(),
                'registration_complete': photo_num == REQUIRED_PHOTO_COUNT,
                'photo_remain': REQUIRED_PHOTO_COUNT - photo_num
            },
        )


    @has_permission(
        Permission.profile_edit,
        context_factory=_context_factory
    )
    async def delete_profile(
        request: web.Request,
        context: ProfileContext
    ) -> web.Response:

        profile = context.profile

        if not profile:
            raise make_api_response(
                web.HTTPNotFound,
                body={
                    'error': {
                        'msg': f'Profile profile_id {context.profile} '
                               f'does not exists'
                    },
                    'data': {},
                },
            )

        log.info('Deleting profile.')

        await _delete_profile(request.app, context)
```

```
    exist_flag = bool(await Profile.find_one(
        {
            'profile_id': context.profile_id,
            'company_id': context.company_id
        }
    ))

    return make_api_response(
        web.HTTPOk,
        body={
            'exist': exist_flag
        },
    )


async def profile_list(
    request: web.Request
) -> web.Response:
    headers = request.headers
    company_id = headers.get('company_id')
    try:
        company = await Company.find_one(
            {'_id': ObjectId(company_id)}
        )
    except Exception as e:
        raise make_api_response(
            web.HTTPNotFound,
            body={
                'error': {
                    'msg': str(e)
                },
                'data': {},
            },
        )
    if not company:
        raise make_api_response(
            web.HTTPNotFound,
            body={
                'error': {
                    'msg': f'Company _id {company_id} does not exists'
                },
                'data': {},
            },
        )
    token = headers.get('token')
    context = ProfileContext(
        profile_id=None,
        profile=None,
        company_id=company_id,
        token=token,
        company=company
    )
    await check_permission(request, Permission.profile_edit, context=context)

    profile_lst = Profile.find(
            {'company_id': company_id}
    )
    profiles = [
        profile.dump()
        async for profile in profile_lst
    ]

    return make_api_response(
        web.HTTPOk,
        body={
            'dataList': profiles,
```

```
            },
        )


    @has_permission(
        Permission.profile_edit,
        context_factory=_context_factory
    )
    async def auth_profile(
        request: web.Request,
        context: ProfileContext
    ) -> web.Response:
        log.info('Auth profile.')
        profile = context.profile

        if not profile:
            raise make_api_response(
                web.HTTPNotFound,
                body={
                    'error': {
                        'msg': f'Profile profile_id {context.profile_id} '
                               f'does not exists'
                    },
                    'data': {},
                },
            )

        if not profile.active:
            raise make_api_response(
                web.HTTPForbidden,
                body={
                    'error': {
                        'msg': f'Profile profile_id {context.profile.profile_id} '
                               f'not fully registered'
                    },
                    'data': {},
                },
            )

        if request.headers.get('client'):
            try:
                photo_bytes = await request.read()
            except Exception as e:
                return make_api_response(web.HTTPUnprocessableEntity, text=str(e))
        else:
            try:
                reader = await request.multipart()
            except Exception as e:
                return make_api_response(web.HTTPUnprocessableEntity, text=str(e))

            image = await reader.next()
            photo_bytes = await image.read(decode=True)

        valid_photo, errors = await validate_photo(
            request.app,
            photo_bytes,
            registration=False
        )

        if not valid_photo:
            return make_api_response(
                web.HTTPUnprocessableEntity,
                body={
                    'errors': errors,
                },
            )
        photo_name = await save_auth_photo_to_s3(request.app, context, photo_bytes)
```

```
    confirmed = await _auth_profile(request.app, context, photo_name)

    if not confirmed:
        return make_api_response(
            web.HTTPUnauthorized,
            body={
                'data': f'Company {profile.company_id} '
                        f'Profile: {profile.profile_id} NOT authenticated'
            },
        )

    return make_api_response(
        web.HTTPOk,
        body={
            'data': f'Company {profile.company_id} '
                    f'Profile: {profile.profile_id} AUTHENTICATED'
        },
    )
```

faceapi/app/forms/company.py

```
from marshmallow import (
    Schema,
    fields,
    validate,
)

MAX_COMPANY_NAME_LEN = 256
MAX_INFO_LEN = 100
error_max_length = 'Кількість символів повинна бути не більша за {max}'
error_common_required = "Це поле обов'язкове"
error_email = "Неправильний формат електронної адреси"


class CompanySchema(Schema):
    name = fields.Str(
        validate=validate.Length(
            max=MAX_COMPANY_NAME_LEN,
            error=error_max_length
        )
    )

    email = fields.Email(error_messages={'invalid': error_email})

    information = fields.Str(
        validate=validate.Length(
            max=MAX_INFO_LEN,
            error=error_max_length
        )
    )
```

faceapi/app/forms/profile.py

```
from marshmallow import (
    Schema,
    fields,
    validates,
)

class _Photo(Schema):
    face_id = fields.Str(required=True)
    url = fields.Str(required=True)


class ProfileSchema(Schema):
    photos = fields.Nested(_Photo, many=True)
```

```
    main_photo = fields.Nested(_Photo, many=False)

    @validates('photos')
    def _validate_photo_amount(self, data: fields.Nested):
        return True
```

faceapi/app/bl/profile.py

```
import logging
import time

from app.bl.action import log_action
from app.const import S3_BUCKET, REQUIRED_PHOTO_COUNT, S3_BUCKET_AUTH
from app.models.photo import Photo
from app.models.profile import Profile
from app.services import (
    s3,
    rekognition
)
from app.utils.common import collection_name

log = logging.getLogger(__name__)


def photo_count(profile):
    return len(profile.photos)


async def create_new_profile(app, context, photo_bytes):
    profile = Profile(
        profile_id=context.profile_id,
        company_id=context.company_id
    )

    photo_name = f'{profile.company_id}_{profile.profile_id}_photo_' \
                 f'{photo_count(profile)}'

    await s3.put_photo(
        app=app,
        bucket=S3_BUCKET,
        photo_bytes=photo_bytes,
        photo_name=photo_name
    )

    collection_id = collection_name(profile.company_id)
    face_id = await rekognition.add_face(
        app=app,
        bucket=S3_BUCKET,
        photo_name=photo_name,
        collection_id=collection_id
    )

    photo = Photo(
        face_id=face_id,
        url=f'{S3_BUCKET}/{photo_name}'
    )
    profile.photos = [photo]
    profile.main_photo = photo

    await profile.commit()
    await log_action(
        action=f'створено новий профіль',
        company_id=profile.company_id,
        profile_id=profile.profile_id
    )

    return profile
```

```python
async def add_photo_to_profile(app, context, photo_bytes):
    profile = context.profile

    photo_name = f'{profile.company_id}_{profile.profile_id}_photo_' \
                 f'{photo_count(profile)}'

    await s3.put_photo(
        app=app,
        bucket=S3_BUCKET,
        photo_bytes=photo_bytes,
        photo_name=photo_name
    )

    collection_id = collection_name(profile.company_id)
    face_id = await rekognition.add_face(
        app=app,
        bucket=S3_BUCKET,
        photo_name=photo_name,
        collection_id=collection_id
    )

    photo = Photo(
        face_id=face_id,
        url=f'{S3_BUCKET}/{photo_name}'
    )
    profile.photos.append(photo)

    if photo_count(profile) == REQUIRED_PHOTO_COUNT:
        profile.activate()

    await profile.commit()

    await log_action(
        action=f'додано фото до профілю',
        company_id=profile.company_id,
        profile_id=profile.profile_id
    )

    return profile


async def auth_profile(app, context, photo_name):
    profile = context.profile
    collection_id = collection_name(profile.company_id)

    await log_action(
        action=f'автентифікація профілю',
        company_id=profile.company_id,
        profile_id=profile.profile_id
    )

    try:
        faces = await rekognition.search_by_image(
            app=app,
            collection_id=collection_id,
            bucket=S3_BUCKET_AUTH,
            photo_name=photo_name,
        )
    except Exception as e:
        log.error(f'Error while auth profile {profile.profile_id}: {str(e)}')
        return False

    if not faces:
        return False

    matched_face_id = faces[0]['Face']['FaceId']
```

```
    profile_face_ids = [photo.face_id for photo in profile.photos]

    if matched_face_id not in profile_face_ids:
        return False

    return True


async def delete_profile(app, context):
    profile = context.profile

    for photo in profile.photos:
        bucket, photo_name = photo.url.split('/')
        await s3.delete_photo(app, bucket=bucket, photo_name=photo_name)

        collection_id = collection_name(profile.company_id)
        await rekognition.delete_face(
            app=app,
            face_id=photo.face_id,
            collection_id=collection_id
        )

    await log_action(
        action=f'видалення профілю',
        company_id=profile.company_id,
        profile_id=profile.profile_id
    )

    await profile.delete()


async def save_auth_photo_to_s3(app, context, photo_bytes):
    profile = context.profile
    curr_time = time.ctime(time.time()).replace(' ', '_')
    photo_name = f'{profile.company_id}_{profile.profile_id}_auth_' \
                 f'photo_{curr_time}'

    await s3.put_photo(
        app=app,
        bucket=S3_BUCKET_AUTH,
        photo_bytes=photo_bytes,
        photo_name=photo_name
    )

    return photo_name
```

faceapi/app/bl/action.py

```
from app.models.action import Action
from app.utils.common import gen_now

import logging

log = logging.getLogger(__name__)

async def log_action(action, company_id=None, profile_id=None):
    dt_now = gen_now()
    if not company_id:
        company_id = 'admin'

    data = {
        'company_id': company_id,
        'action': action,
        'datetime': dt_now,
    }

    if profile_id:
        data['profile_id'] = profile_id
```

```
    action_obj = Action(**data)
    await action_obj.commit()
```

faceapi_client/app/__main__.py

```
import logging

import aiohttp_autoreload
from aiohttp import web

from .application import create_app

log = logging.getLogger(__name__)


if __name__ == '__main__':
    # Create app
    app = create_app()

    app_config = app['config']['app']

    if app_config['debug']:
        aiohttp_autoreload.start()

    log.info('Listen on {}:{}'.format(app_config['host'], app_config['port']))

    web.run_app(app, port=app_config['port'])
```

faceapi_client/app/application.py

```
import base64
import logging
import jinja2

from aiohttp import web
from aiohttp_jinja2 import setup as jinja_setup
from aiohttp_session import (
    get_session,
    setup as session_setup,
)
from aiohttp_session.cookie_storage import EncryptedCookieStorage

from app import signals
from app.config import get_config
from app.const import APP_DIR
from app.routes import setup_routes

log = logging.getLogger(__name__)


async def current_user_ctx_processor(request):
    session = await get_session(request)
    is_anonymous = True
    if 'user' in session:
        is_anonymous = False

    return dict(current_user=not is_anonymous, is_anonymous=is_anonymous)


@web.middleware
async def user_session_middleware(request, handler):
    request.session = await get_session(request)
    response = await handler(request)
    return response


def setup_middlewares(app):
```

```
    app.middlewares.append(user_session_middleware)


def create_app() -> web.Application:
    """create application"""
    config = get_config()

    # App configuration
    app_config = config['app']

    # Create app
    app = web.Application(
        debug=app_config['debug']
    )

    app['config'] = config

    key = b'TyzLMReLCWUiPsTFMActw_0dtEU7kAcFXHNYYm64DNI='
    secret_key = base64.urlsafe_b64decode(key)
    session_setup(app, EncryptedCookieStorage(secret_key))

    # logging
    logging.basicConfig(level=logging.INFO)

    # setup jinja2
    jinja_setup(
        app,
        loader=jinja2.FileSystemLoader(str(APP_DIR / 'templates')),
        context_processors=[current_user_ctx_processor]
    )
    log.info('Jinja2 initialized')

    # Setup signals
    # To run on app startup
    app.on_startup.extend(
        [
            signals.init_client_session,
            signals.init_faceapi,
        ]
    )

    # To run before shutdown
    app.on_cleanup.extend(
        [
            signals.destroy_client_session,
        ]
    )

    # Routes
    setup_routes(app)

    setup_middlewares(app)

    return app
```

faceapi_client/app/routes.py

```
from aiohttp import web

from app.handlers.company import (
    regenerate_token,
    delete,
    edit as edit_handler,
    add as add_handler
)
from app.handlers.login import login as login_post
from app.handlers.login import logout as logout_post
from app.handlers.profile import register, auth
```

```
from app.views.index import index
from app.views.company import (
    company_list_view,
    add as add_view,
    edit as edit_view
)
from app.views.demo import demo_view
from app.views.history import history_view
from app.views.login import login


def setup_routes(app: web.Application) -> None:
    # interface
    app.router.add_get('/', index)
    app.router.add_get('/login', login)
    app.router.add_post('/login', login_post)
    app.router.add_get('/logout', logout_post)

    app.router.add_get('/companies', company_list_view)
    app.router.add_get('/companies/add', add_view)
    app.router.add_get('/companies/{company_id}/edit', edit_view)

    app.router.add_get('/history', history_view)

    app.router.add_get('/demo', demo_view)

    # handlers: START
    # companies
    app.router.add_post('/companies/{company_id}/token', regenerate_token)
    app.router.add_post('/companies/{company_id}/delete', delete)
    app.router.add_post('/companies/add', add_handler)
    app.router.add_post('/companies/{company_id}/edit', edit_handler)

    # profiles
    app.router.add_post('/profile/register', register)
    app.router.add_post('/profile/auth', auth)
    # handlers: END
```

faceapi_client/app/security.py

```
import inspect
import logging
from typing import Any, Callable, Dict, Optional

from aiohttp import web
from aiohttp_session import get_session

from app.const import Permission

log = logging.getLogger(__name__)

async def check_permission(request, permission, context=None):
    allowed = await permits(request, permission)
    if not allowed:
        raise web.HTTPForbidden()


async def permits(
    request: web.BaseRequest, permission: Permission, context: Any = None
) -> bool:
    if permission == Permission.admin:
        return await _permits_admin(request, context)
    else:
        raise ValueError(f'No algo for {permission} defined')


async def _permits_admin(
    request: web.BaseRequest,
```

```
    context
) -> bool:
    session = await get_session(request)
    if 'user' not in session:
        return False
    return True
```

faceapi_client/app/signals.py

```
import logging

from aiohttp import web, ClientSession

from app.services import faceapi

log = logging.getLogger(__name__)


async def init_client_session(app: web.Application) -> None:
    app['session'] = ClientSession()
    log.info('ClientSession initialized')


async def init_faceapi(app: web.Application) -> None:
    app['faceapi'] = faceapi.FaceAPIManager(app)
    log.info('FaceAPI initialized')


async def destroy_client_session(app: web.Application) -> None:
    session = app['session']
    await session.close()
    log.info('ClientSession closed')
```

faceapi_client/app/views/company.py

```
import typing as t

import aiohttp_jinja2

from aiohttp import web
from app.const import Permission
from app.security import check_permission


@aiohttp_jinja2.template('company_edit.html')
async def add(
    request: web.Request
) -> t.Any:
    try:
        await check_permission(request, Permission.admin)
    except Exception:
        return web.HTTPFound('/')

    return {'company': {}, 'error': {}, 'edit': False}


@aiohttp_jinja2.template('company_edit.html')
async def edit(
    request: web.Request
) -> t.Any:
    try:
        await check_permission(request, Permission.admin)
    except Exception:
        return web.HTTPFound('/')
    company_id = request.match_info['company_id']
    app = request.app

    company = await app['faceapi'].get_company(company_id)
```

```
    return {'company': company, 'error': {}, 'edit': True}


@aiohttp_jinja2.template('company.html')
async def company_list_view(
    request: web.Request
) -> t.Any:
    try:
        await check_permission(request, Permission.admin)
    except Exception:
        return web.HTTPFound('/')
    app = request.app

    companies = await app['faceapi'].get_company_list()

    return {'companies': companies}
```

faceapi_client/app/views/demo.py

```
import typing as t

import aiohttp_jinja2

from aiohttp import web


@aiohttp_jinja2.template('demo.html')
async def demo_view(
    request: web.Request
) -> t.Any:

    return {
        'reg_data': {},
        'reg_response': {},
        'auth_data': {},
        'auth_response': {}
    }
```

faceapi_client/app/views/history.py

```
import typing as t

import aiohttp_jinja2

from aiohttp import web
from app.const import Permission
from app.security import check_permission


@aiohttp_jinja2.template('history.html')
async def history_view(
    request: web.Request
) -> t.Any:
    try:
        await check_permission(request, Permission.admin)
    except Exception:
        return web.HTTPFound('/')
    app = request.app

    actions = await app['faceapi'].get_action_list()

    return {'actions': actions}
```

faceapi_client/app/views/index.py

```
import aiohttp_jinja2
```

```
from aiohttp import web


@aiohttp_jinja2.template('index.html')
async def index(
    request: web.Request
) -> dict:
    return {}
```

faceapi_client/app/views/login.py

```
import typing as t

import aiohttp_jinja2

from aiohttp import web


@aiohttp_jinja2.template('login.html')
async def login(
    request: web.Request
) -> t.Any:

    return {}
```

faceapi_client/app/services/faceapi.py

```
import logging
import json

from aiohttp import web

log = logging.getLogger(__name__)

class FaceAPIManager:
    def __init__(self, app: web.Application) -> None:
        self.url = app['config']['faceapi']['url']
        self.session = app['session']

    async def get_company_list(self):
        async with self.session.get(f'{self.url}/companies') as response:
            json_body = await response.json()
            companies = json_body['dataList']

        return companies

    async def get_company(self, doc_id):
        async with self.session.get(
                f'{self.url}/companies/{doc_id}'
        ) as response:
            json_body = await response.json()
            company = json_body['data']

        return company

    async def add_company(self, company_data):
        async with self.session.post(
                f'{self.url}/companies', json=company_data
        ) as response:
            json_body = await response.json()

        return json_body

    async def edit_company(self, doc_id, company_data):
        async with self.session.patch(
                f'{self.url}/companies/{doc_id}', json=company_data
        ) as response:
            json_body = await response.json()
```

```
        return json_body

    async def delete_company(self, doc_id):
        async with self.session.delete(
                f'{self.url}/companies/{doc_id}'
        ) as response:
            json_body = await response.json()
            activated = json_body['activated']
        return activated

    async def regenerate_company_token(self, doc_id):
        async with self.session.patch(
                f'{self.url}/companies/{doc_id}/token'
        ) as response:
            json_body = await response.json()
            token = json_body['token']

        return token

    async def register_profile(self, company_id, profile_id, token, photo):
        photo_file = photo.file
        headers = {
            'company_id': company_id,
            'token': token,
            'Content-Type': 'multipart/form-data',
            'client': 'faceapi_client',
        }

        async with self.session.patch(
            f'{self.url}/profiles/{profile_id}',
            data=photo_file,
            headers=headers
        ) as response:
            if response.status == 403:
                json_body = {'error': 'Доступ заборонено'}
            else:
                json_body = await response.json()

        return json.dumps(json_body, indent=2, ensure_ascii=False)

    async def auth_profile(self, company_id, profile_id, token, photo):
        photo_file = photo.file
        headers = {
            'company_id': company_id,
            'token': token,
            'Content-Type': 'multipart/form-data',
            'client': 'faceapi_client',
        }
        async with self.session.post(
            f'{self.url}/profiles/{profile_id}/auth',
            data=photo_file,
            headers=headers
        ) as response:
            if response.status == 403:
                json_body = {'error': 'Доступ заборонено'}
            else:
                json_body = await response.json()

        return json.dumps(json_body, indent=2, ensure_ascii=False)

    async def get_action_list(self):
        async with self.session.get(f'{self.url}/actions') as response:
            json_body = await response.json()
            actions = json_body.get('dataList')

        return actions
```

faceapi_client/app/handlers/company.py

```python
import logging

import aiohttp_jinja2
from aiohttp import web

from app.const import Permission
from app.security import check_permission

log = logging.getLogger(__name__)


async def regenerate_token(request: web.Request):
    try:
        await check_permission(request, Permission.admin)
    except Exception:
        return web.HTTPFound('/')
    company_id = request.match_info['company_id']
    app = request.app
    await app['faceapi'].regenerate_company_token(company_id)

    return web.HTTPFound('/companies')


async def delete(request: web.Request):
    try:
        await check_permission(request, Permission.admin)
    except Exception:
        return web.HTTPFound('/')
    company_id = request.match_info['company_id']
    app = request.app
    await app['faceapi'].delete_company(company_id)

    return web.HTTPFound('/companies')


async def add(request: web.Request):
    try:
        await check_permission(request, Permission.admin)
    except Exception:
        return web.HTTPFound('/')
    data = await request.post()
    company_data = {
        'name': data.get('name', ''),
        'email': data.get('email', ''),
        'information': data.get('information', ''),
    }
    app = request.app
    resp = await app['faceapi'].add_company(company_data)
    company = resp.get('data')
    errors = resp.get('error')
    if not company or errors:
        return aiohttp_jinja2.render_template(
            'company_edit.html',
            request,
            {'company': company_data, 'errors': errors, 'edit': False}
        )

    return web.HTTPFound('/companies')


async def edit(request: web.Request):
    try:
        await check_permission(request, Permission.admin)
    except Exception:
        return web.HTTPFound('/')
```

```
    company_id = request.match_info['company_id']
    data = await request.post()
    company_data = {
        'id': company_id,
        'name': data.get('name', ''),
        'email': data.get('email', ''),
        'information': data.get('information', ''),
    }
    app = request.app
    resp = await app['faceapi'].edit_company(company_id, company_data)
    company = resp.get('data')
    errors = resp.get('error')
    if not company or errors:
        return aiohttp_jinja2.render_template(
            'company_edit.html',
            request,
            {'company': company_data, 'errors': errors, 'edit': True}
        )
    return web.HTTPFound('/companies')
```

faceapi_client/app/handlers/login.py

```
import aiohttp_jinja2
import logging

from aiohttp import web
from aiohttp_session import get_session

log = logging.getLogger(__name__)


async def login(request: web.Request):
    data = await request.post()
    login_val = data['login']
    password = data['password']

    if not login_val == 'admin' or not password == 'admin':
        return aiohttp_jinja2.render_template(
            'login.html',
            request,
            {'errors': 'Неправильні дані для входу.'}
        )

    session = await get_session(request)
    session['user'] = {'login': login_val}

    return web.HTTPFound('/companies')


async def logout(request: web.Request):
    session = await get_session(request)
    del session['user']

    return web.HTTPFound('/')
```

faceapi_client/app/handlers/profile.py

```
import aiohttp_jinja2
import logging

from aiohttp import web

log = logging.getLogger(__name__)


@aiohttp_jinja2.template('demo.html')
async def register(request: web.Request):
    data = await request.post()
```

```
    company_id = data['company_id']
    profile_id = data['profile_id']
    token = data['token']
    photo = data['photo']
    app = request.app
    response = await app['faceapi'].register_profile(
        company_id=company_id,
        profile_id=profile_id,
        token=token,
        photo=photo
    )
    reg_data = {
        'company_id': company_id,
        'profile_id': profile_id,
        'token': token
    }

    return {
        'reg_data': reg_data,
        'reg_response': response,
        'auth_data': {},
        'auth_response': {}
    }


@aiohttp_jinja2.template('demo.html')
async def auth(request: web.Request):
    data = await request.post()
    company_id = data['company_id']
    profile_id = data['profile_id']
    token = data['token']
    photo = data['photo']
    app = request.app
    response = await app['faceapi'].auth_profile(
        company_id=company_id,
        profile_id=profile_id,
        token=token,
        photo=photo
    )
    auth_data = {
        'company_id': company_id,
        'profile_id': profile_id,
        'token': token
    }

    return {
        'reg_data': {},
        'reg_response': {},
        'auth_data': auth_data,
        'auth_response': response
    }
```

faceapi_client/app/templates/base.html

```
<!DOCTYPE html>
<html>
  <head>
      <meta charset="UTF-8">
      <title>{{title}}</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
           <link   href="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css"
rel="stylesheet" media="screen">
      <style type="text/css">
      .container {
          max-width: 1200px;
          padding-top: 10px;
      }
      thead, td, tr, th, tbody  {
```

```
                border: 2px solid gray;
        }
        .table>tbody>tr>td {
            vertical-align: middle;
            border: 2px solid gray
        }

        </style>
    </head>
    <body>
        <nav class="navbar navbar-inverse" role="navigation">
        <div class="container-fluid">
          <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-
example-navbar-collapse-1">
              <span class="sr-only">Toggle navigation</span>
              <span class="icon-bar"></span>
              <span class="icon-bar"></span>
              <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="/">FaceAPI</a>
          </div>

          <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
              {% if current_user %}
                  <ul class="nav navbar-nav">
                      <li><a href="/companies">Організації</a></li>
                      <li><a href="/history">Історія дій</a></li>
                  </ul>
              {% endif %}
              <ul class="nav navbar-nav navbar-right">
                  <li><a href="/demo">FaceAPI DEMO</a></li>
                  {% if current_user %}
                      <li><a href="/logout">Вихід</a></li>
                  {% else %}
                      <li><a href="/login">Вхід</a></li>
                  {% endif %}
              </ul>
          </div>
        </div>
      </nav>
{#    <div>#}
      <div class="container">
        {% block content %}{% endblock %}
      </div>
      <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
      <script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/js/bootstrap.min.js"></script>
    </body>
</html>
```

faceapi_client/app/templates/company.html

```
{% extends "base.html" %}

{% set title = "Список організацій" %}

{% block content %}
    <h1 class="text-center">Список організацій</h1>
    <form action="/companies/add" method="get">
            <button class="btn btn-lg btn-primary btn-block pull-left" style="width: 25%"
type="submit">Додати нову організацію</button>
    </form>
    <br><br><br>
    <div class="container-fluid">
      <table class="table table-bordered">
        <thead class="thead-dark">
            <tr>
                <th class="text-center">Ідентифікатор</th>
```

```
                <th class="text-center">Назва</th>
                <th class="text-center">Email</th>
                <th class="text-center">Додаткова інформація</th>
                <th class="text-center" colspan="2">Ключ доступу</th>
                <th class="text-center">Керування</th>
            </tr>
        </thead>
        <tbody>
            {% for company in companies %}
                {% set company_id = company.get('id') %}
                {% set information = company.get('information') %}
                <tr >
                    <td rowspan="2">{{ company_id }}</td>
                    <td rowspan="2">{{company.get('name')}}</td>
                    <td rowspan="2">{{company.get('email')}}</td>
                    {% if information %}
                        <td rowspan="2">{{ information }}</td>
                    {% else %}
                        <td rowspan="2">Додаткова інформація відсутня</td>
                    {% endif %}
                    <td rowspan="2">{{company.get('token')}}</td>
                    <td rowspan="2">
                        <form action="/companies/{{ company_id }}/token" method="post">
                                        <button  class="btn  btn-primary"  style="width:  100%"
type="submit">Згенерувати новий</button>
                        </form>
                    </td>
                    <td>
                        <form action="/companies/{{ company_id }}/edit" method="get">
                                        <button  class="btn  btn-primary"  style="width:  100%"
type="submit">Редагувати</button>
                        </form>
                    </td>
                </tr>
                <tr>
                    <td>
                        <form action="/companies/{{ company_id }}/delete" method="post">
                                         <button  class="btn  btn-danger"  style="width:  100%"
type="submit">Видалити</button>
                        </form>
                    </td>
                </tr>
            {% endfor %}
        </tbody>
      </table>
    </div>

{% endblock %}
```

faceapi_client/app/templates/company_edit.html

```
{% extends "base.html" %}

{% if company and edit %}
    {% set title = "Редагування організації" %}
{% else %}
    {% set title = "Нова організація" %}
{% endif %}

{% block content %}
    <div class="container" style="max-width: 600px">
        <h2 class="text-center">{{ title }}</h2>
        {% if company and edit %}
            <form action="/companies/{{ company.get('id') }}/edit" method="post">
              <div class="form-group row">
               <label for="input_name" class="col-sm-2 col-form-label">Назва організації</label>
                <div class="col-sm-10">
```

```
                    <input type="text" class="form-control" id="input_name" placeholder="Назва
організації" name="name" value="{{ company.get('name') }}" required>
                </div>
                    {% if errors %}
                     <div class="col-sm-10">
                         <p style="color:red;">{{ errors.get('name', '') }}</p>
                     </div>
                {% endif %}
              </div>
              <div class="form-group row">
                            <label   for="input_email"   class="col-sm-2   col-form-label">Email
організації</label>
                <div class="col-sm-10">
                   <input type="text" class="form-control" id="input_email" placeholder="Email
організації" name="email" value="{{ company.get('email') }}" required>
                </div>
                  {% if errors %}
                  <div class="col-sm-10">
                     <p style="color:red;">{{ errors.get('email', '') }}</p>
                  </div>
                {% endif %}
              </div>
              <div class="form-group row">
                    <label  for="input_information"  class="col-sm-2  col-form-label">Додаткова
інформація</label>
                <div class="col-sm-10">
                             <input  type="text"  class="form-control"  id="input_information"
placeholder="Додаткова інформація" name="information" value="{{ company.get('information') }}">
                </div>
                  {% if errors %}
                  <div class="col-sm-10">
                     <p style="color:red;">{{ errors.get('information', '') }}</p>
                  </div>
                {% endif %}
              </div>
              <br>
              <button class="btn btn-lg btn-success btn-block" type="submit">Зберегти</button>
            </form>
        {% elif company %}
            <form action="/companies/add" method="post">
              <div class="form-group row">
               <label for="input_name" class="col-sm-2 col-form-label">Назва організації</label>
                <div class="col-sm-10">
                    <input type="text" class="form-control" id="input_name" placeholder="Назва
організації" name="name" value="{{ company.get('name') }}" required>
                </div>
                    {% if errors %}
                     <div class="col-sm-10">
                         <p style="color:red;">{{ errors.get('name', '') }}</p>
                     </div>
                {% endif %}
              </div>
              <div class="form-group row">
                            <label   for="input_email"   class="col-sm-2   col-form-label">Email
організації</label>
                <div class="col-sm-10">
                   <input type="text" class="form-control" id="input_email" placeholder="Email
організації" name="email" value="{{ company.get('email') }}" required>
                </div>
                  {% if errors %}
                  <div class="col-sm-10">
                     <p style="color:red;">{{ errors.get('email', '') }}</p>
                  </div>
                {% endif %}
              </div>
              <div class="form-group row">
```

```
                    <label for="input_information" class="col-sm-2 col-form-label">Додаткова інформація</label>
                <div class="col-sm-10">
                            <input type="text" class="form-control" id="input_information" placeholder="Додаткова інформація" name="information" value="{{ company.get('information') }}">
                </div>
                  {% if errors %}
                  <div class="col-sm-10">
                     <p style="color:red;">{{ errors.get('information', '') }}</p>
                  </div>
                {% endif %}
              </div>
              <br>
              <button class="btn btn-lg btn-success btn-block" type="submit">Зберегти</button>
            </form>
        {% else %}
            <form action="/companies/add" method="post">
              <div class="form-group row">
               <label for="input_name" class="col-sm-2 col-form-label">Назва організації</label>
                <div class="col-sm-10">
                    <input type="text" class="form-control" id="input_name" placeholder="Назва організації" name="name" required>
                </div>
                {% if errors %}
                    <div class="col-sm-10">
                        <p style="color:red;">{{ errors.get('name', '') }}</p>
                    </div>
                {% endif %}
              </div>
              <div class="form-group row">
                        <label for="input_email" class="col-sm-2 col-form-label">Email організації</label>
                <div class="col-sm-10">
                   <input type="text" class="form-control" id="input_email" placeholder="Email організації" name="email" required>
                </div>
                {% if errors %}
                  <div class="col-sm-10">
                     <p style="color:red;">{{ errors.get('email', '') }}</p>
                  </div>
                {% endif %}
              </div>
              <div class="form-group row">
                    <label for="input_information" class="col-sm-2 col-form-label">Додаткова інформація</label>
                <div class="col-sm-10">
                            <input type="text" class="form-control" id="input_information" placeholder="Додаткова інформація" name="information">
                </div>
                {% if errors %}
                  <div class="col-sm-10">
                     <p style="color:red;">{{ errors.get('information', '') }}</p>
                  </div>
                {% endif %}
              </div>
              <br>
              <button class="btn btn-lg btn-success btn-block" type="submit">Зберегти</button>
            </form>
        {% endif %}
    </div>
{% endblock %}
```

faceapi_client/app/templates/demo.html

```
{% extends "base.html" %}

{% set title = "FaceAPI DEMO" %}
```

```
{% block content %}
    <div class="container" style="max-width: 600px">
        <h2 class="text-center">ЗАПИТ НА РЕЄСТРАЦІЮ</h2>
        <form action="/profile/register" method="post" enctype="multipart/form-data">
          <div class="form-group row">
           <label for="input_company_id" class="col-sm-2 col-form-label">ID організації</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="input_company_id" placeholder="ID
організації" name="company_id" value="{{ reg_data.get('company_id', '') }}" required>
            </div>

          </div>
          <div class="form-group row">
            <label for="input_token" class="col-sm-2 col-form-label">Ключ доступу</label>
            <div class="col-sm-10">
             <input type="text" class="form-control" id="input_token" placeholder="Ключ доступу"
name="token" value="{{ reg_data.get('token', '') }}" required>
            </div>
          </div>
          <div class="form-group row">
            <label for="input_profile_id" class="col-sm-2 col-form-label">ID клієнта</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="input_profile_id" placeholder="ID
клієнта" name="profile_id" value="{{ reg_data.get('profile_id', '') }}" required>
            </div>
          </div>

          <div class="form-group row">
                    <label  class="col-sm-2  custom-file-label"  for="validatedCustomFile">Фото
клієнта</label>
            <div class="col-sm-10">
                                <input   type="file"   class="custom-file-input"   name="photo"
id="validatedCustomFile" required>
            </div>
          </div>

          <div class="form-group row">
            <div class="col-sm-10">
                        <button  type="submit"  class="btn  btn-primary">Відправити  запит  на
реєстрацію</button>
            </div>
          </div>
        </form>
        <h2 class="text-center">ВІДПОВІДЬ ВІД FaceAPI</h2>
        <pre>{{ reg_response }}</pre>
    </div>
    <hr style="width: 100%; color: black; height: 3px; background-color:gray; max-width: 600px"
/>
    <div class="container" style="max-width: 600px">
        <h2 class="text-center">ЗАПИТ НА АВТЕНТИФІКАЦІЮ</h2>
        <form action="/profile/auth" method="post" enctype="multipart/form-data">
             <div class="form-group row">
           <label for="input_company_id" class="col-sm-2 col-form-label">ID організації</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="input_company_id" placeholder="ID
організації" name="company_id" value="{{ auth_data.get('company_id', '') }}" required>
            </div>

          </div>
          <div class="form-group row">
            <label for="input_token" class="col-sm-2 col-form-label">Ключ доступу</label>
            <div class="col-sm-10">
             <input type="text" class="form-control" id="input_token" placeholder="Ключ доступу"
name="token" value="{{ auth_data.get('token', '') }}" required>
            </div>
          </div>
          <div class="form-group row">
```

```
            <label for="input_profile_id" class="col-sm-2 col-form-label">ID клієнта</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" id="input_profile_id" placeholder="ID клієнта" name="profile_id" value="{{ auth_data.get('profile_id', '') }}" required>
            </div>
          </div>

          <div class="form-group row">
                  <label class="col-sm-2 custom-file-label" for="validatedCustomFile">Фото клієнта</label>
            <div class="col-sm-10">
                                <input type="file" class="custom-file-input" name="photo" id="validatedCustomFile" required>
{#                <div class="invalid-feedback">Фото клієнта є обов'язковим</div>#}
            </div>
          </div>

          <div class="form-group row">
            <div class="col-sm-10">
                        <button type="submit" class="btn btn-primary">Відправити запит на автентифікацію</button>
            </div>
          </div>
        </form>
        <h2 class="text-center">ВІДПОВІДЬ ВІД FaceAPI</h2>
        <pre>{{ auth_response }}</pre>
    </div>


{% endblock %}
```

faceapi_client/app/templates/history.html

```
{% extends "base.html" %}

{% set title = "Історія дій в системі" %}

{% block content %}
  <h1 class="text-center">Історія дій в системі</h1>
    <div class="container-fluid">
      <table class="table table-bordered">
        <thead>
            <tr>
                <th>Дата і час</th>
                <th>Дія</th>
                <th>ID Організації</th>
                <th>ID Профілю</th>
            </tr>
        </thead>
        <tbody>
            {% for action in actions %}
                <tr>
                    <td>{{action.get('datetime')}}</td>
                    <td>{{action.get('action')}}</td>
                    <td>{{action.get('company_id')}}</td>
                    <td>{{action.get('profile_id')}}</td>
                </tr>
            {% endfor %}
        </tbody>
      </table>
    </div>
{% endblock %}
```

faceapi_client/app/templates/index.html

```
{% extends "base.html" %}

{% set title = "Інформація" %}
```

```
{% block content %}
    <div align="center" class="container">
        <h1>Підсистема розпізнавання та автентифікації обличь в системі оплати – Система FaceAPI
</h1>
        <h2>Дипломний проект</h2>
        <h2>студента групи ІС-63</h2>
        <h2>кафедри АСОІУ, ФІОТ, КПІ ім. І. Сікорського</h2>
        <h2>Портяного Івана Сергійовича</h2>
    </div>
{% endblock %}
```

faceapi_client/app/templates/login.html

```
{% extends "base.html" %}

{% set title = "Вхід" %}

{% block content %}
    <div class="container" style="max-width: 600px">
      <form class="form-signin" action="/login" method="post" accept-charset="utf-8">
         <h2 class="form-signin-heading" align="center">Авторизація адміністратора в Систему
FaceAPI</h2>
        <label for="inputEmail" class="sr-only">Логін</label>
       <input type="text" id="inputEmail" name="login" class="form-control" placeholder="Логін"
required="" autofocus="">
        <br>
        <label for="inputPassword" class="sr-only">Пароль</label>
            <input  type="password"  id="inputPassword"  class="form-control"  name="password"
placeholder="Пароль" required="">

        {% if errors %}
            <p style="color:red;">{{ errors }}</p>
        {% endif %}
        <br>
        <button class="btn btn-lg btn-primary btn-block" type="submit">Вхід</button>
      </form>
    </div>
{% endblock %}
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації і управління

УЗГОДЖЕНО
Керівник проєкту
_____ *Олена ЖДАНОВА*
(підпис) (вл. ім’я, прізвище)
“13” квітня 2020 р.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)
“14” квітня 2020 р.

ПІДСИСТЕМА РОЗПІЗНАВАННЯ ТА АВТЕНТИФІКАЦІЇ ОБЛИЧЬ В СИСТЕМІ ОПЛАТИ

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 6318.01.000 ТЗ*

на 10 сторінках

Київ – 2020 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ 3

1.1 Повне найменування системи та її умовне позначення 3

1.2 Найменування організації-замовника та організацій-учасників робіт . 3

1.3 Перелік документів, на підставі яких створюється система 3

1.4 Планові терміни початку і закінчення роботи зі створення системи 3

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ 4

2.1 Призначення системи 4

2.2 Цілі створення системи 4

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ 5

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6

4.1 Вимоги до функціональних характеристик 6

4.2 Вимоги до надійності 6

4.3 Вимоги до складу і параметрів технічних засобів 7

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ 8

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ 9

6.1 Види випробувань 9

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повна назва системи**:** Підсистема розпізнавання та автентифікації обличь в системі оплати, далі – **Система FaceAPI**.

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовником є ТОВ «УАПРОМ». Адреса замовника: Україна, 02121, м. Київ, вул. Харківське шосе, буд. 201-203, корпус 2А, літера Ф.

Розробником системи є Портяний Іван Сергійович – студент групи ІС-63 кафедри автоматизованих систем обробки інформації та управління факультету інформатики та обчислювальної техніки Національного технічного університету України "Київський політехнічний інститут ім. Ігоря Сікорського".

1.3 Перелік документів, на підставі яких створюється система

Завдання на переддипломну практику є підставою для розробки Системи FaceAPI.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи по розробці Системи FaceAPI — 29 березня 2020 року.

Плановий строк закінчення роботи по розробці Системи FaceAPI — 11 травня 2020 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення системи

Призначенням Системи FaceAPI є розпізнавання та автентифікація обличчя для підтвердження платежів у системі оплати.

2.2 Цілі створення системи

Цілями розробки Системи FaceAPI є покращити процес оплати за рахунок:

- спрощення процесу підтвердження платежу;
- підвищення рівня безпеки під час проведення платежів.

Для того, щоб досягнути поставлених цілей необхідно реалізувати наступні задачі:

- створення, редагування та видалення організацій-користувачів, які використовуватимуть систему;
- створення, редагування та видалення клієнтів організації для розпізнавання обличчя;
- розпізнавання та автентифікація обличчя по фото.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Для використання Системи FaceAPI потрібно зареєструвати свою організацію в системі та отримати в адміністратора системи ключ доступу для подальшого використання. Після реєстрації організації надається можливість використовувати Систему FaceAPI, а саме:

- додавати профілі своїх клієнтів та їх фото;
- редагувати фото профілів;
- видаляти фото профілів;
- переглядати список всіх доданих профілів;
- надсилати фото для визначення обличчя та автентифікації.

Для того, щоб мати можливість керувати зареєстрованими організаціями потрібно авторизуватись як адміністратор. Адміністратор виконує наступні функції:

- реєстрація, редагування та видалення нових організацій;
- перегляд історії дій в системі;
- створення нових ключів доступу для організацій.

Об’єктом автоматизації є сам процес підтвердження платежу під час проведення оплати.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Система FaceAPI повинна створити умови для швидкого підтвердження платежу за допомогою фото обличчя. Набір функцій, які повинна реалізувати Система FaceAPI можна поділити на дві групи: функції для організації-користувача та функції для організатора.

Система FaceAPI має реалізувати наступні функції для адміністратора:

- функція додавання, редагування та видалення організації;
- функція перегляду історії дій в системі.

Система FaceAPI має реалізувати наступні функції для організації:

- функція додавання, редагування та видалення клієнта;
- функція автентифікації обличчя;
- функція доступу організації до профілів зареєстрованих клієнтів по ключу.

4.2 Вимоги до надійності

Система FaceAPI повинна безперебійно функціонувати та забезпечувати відновлення необхідних для роботи функцій при виникненні непередбачуваних ситуацій, наприклад: раптове перезавантаження системи, перезапуск серверу.

У випадках, коли Система FaceAPI не має можливості обробити запит або під час її роботи виникають помилки, користувачу повинне надсилатись повідомлення з описом проблеми.

Система FaceAPI повинна забезпечувати конфіденційність фото профілів клієнтів та надавати доступ до них тільки організаціям-власникам цих профілів за визначеним ключем доступу.

Будь-які збої роботи Системи FaceAPI не повинні порушувати цілісність даних в інформаційних сховищах.

4.3 Вимоги до складу і параметрів технічних засобів

Для надання можливості запуску та функціонування Системи FaceAPI необхідно забезпечити наявність серверу з наступними вимогами:

- оперативна пам’ять – не менше 2 Гб;
- жорсткий диск – 50 Гб;
- процесор – чотирьохядерний процесор з частотою 2.1 Гц;
- операційна система Ubuntu 16.04;
- встановлений Docker.

Для можливості працювати з Системою FaceAPI (реєструвати нові профілі клієнтів, проводити автентифікацію обличчя) потрібно мати доступ до мережі інтернет та мати можливість відправляти HTTP-запити.

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Нижче наведено основні етапи виконання робіт з розробки Системи FaceAPI.

№ п/п	Назва етапу роботи	Термін виконання етапу	Результат виконання
1.	Підготовка технічного завдання на розробку програмного продукту	13.04.2020	
2.	Розробка сценарію роботи	15.04.2020	
3.	Технічне проектування – функціональність, модулі, задачі, цілі тощо	16.04.2020	
4.	Узгодження з керівником інтерфейсу користувача	07.04.2020	
5.	Розробка інформаційного забезпечення	22.04.2020	
6.	Розробка програмного забезпечення	29.04.2020	
7.	Налагодження програми	09.05.2020	
8.	Тестування програми	10.05.2020	
9.	Здача готового програмного продукту замовника	12.05.2020	

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Для того, щоб перевірити правильність роботи Системи FaceAPI будуть проведені випробування на основі таких документів:

- ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

Випробування повинні проводитись не пізніше 17.05.2020 на комп’ютері замовника. На випробування мають бути пред’явлені наступні документи:

- програма та методика випробувань;
- керівництво користувача;
- технічне завдання.

6.1 Види випробувань

Для перевірки правильності роботи Системи FaceAPI буде проведено наступні етапи випробувань:

- ознайомлення;
- функціональне тестування;
- тестування збереження цілісності даних.

На етапі ознайомлення проводиться перевірка наявності усієї програмної документації, перевірка складу програмних засобів, а також перевірка належності технічних характеристик Системи FaceAPI.

На етапі функціонального тестування проводиться перевірка функціоналу Системи FaceAPI на відповідність до вимог. Буде проведено тестування наступних функцій:

а) зі сторони адміністратору:
   1) авторизація;
   2) створення, редагування та видалення організацій-користувачів системи;
   3) генерування нового ключа доступу для організацій-користувачів;
   4) перегляд історії дій в системі.

б) зі сторони організації-користувача:
   1) додавання, редагування та видалення клієнтів для розпізнавання обличчя;
   2) відправка фото обличчя для автентифікації.

Також передбачується тестування Системи FaceAPI на виведення повідомлень при введені некоректних даних та на виведення відповідних повідомлень при виникнення помилок в роботі.

На етапі перевірки збереження цілісності даних буде проведено перевірку збереження даних в інформаційних сховищах при непередбачуваному перериванні роботи Системи FaceAPI.

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

ID перевірки:
1003881980

Дата перевірки:
08.06.2020 18:22:35 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2020 16:42:59 EEST

ID користувача:
77149

Назва документу: **Portjanij_bachelor_is63**

ID файлу: **1003896804** Кількість сторінок: **65** Кількість слів: **10085** Кількість символів: **74674** Розмір файлу: **1.72 MB**

8.22% Схожість

Найбільша схожість: **2.88%** з джерело бібліотеки. ID файлу: **1003896820**

5.28% Схожість з Інтернет джерелами	102	Page 67
6.4% Текстові збіги по Бібліотеці акаунту	350	Page 68

0.16% Цитат

Цитати	1	Page 69

Вилучення переліку посилань вимкнено

0% Вилучень

Вилучений текст відсутній

Підміна символів

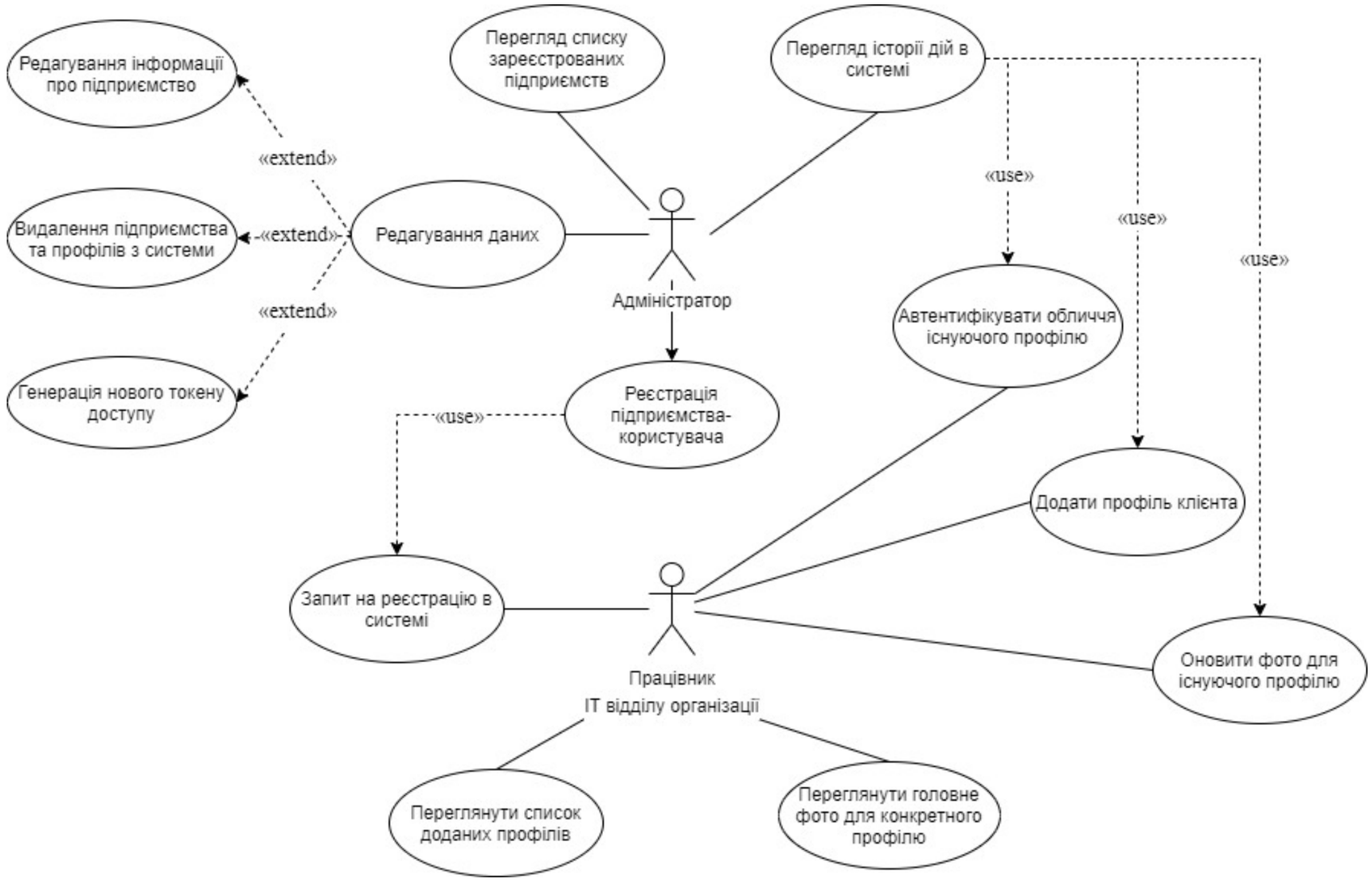
Заміна символів	3

Графічний матеріал
до дипломного проєкту

на тему: Підсистема розпізнавання та автентифікації обличь в системі оплати

Київ – 2020 року

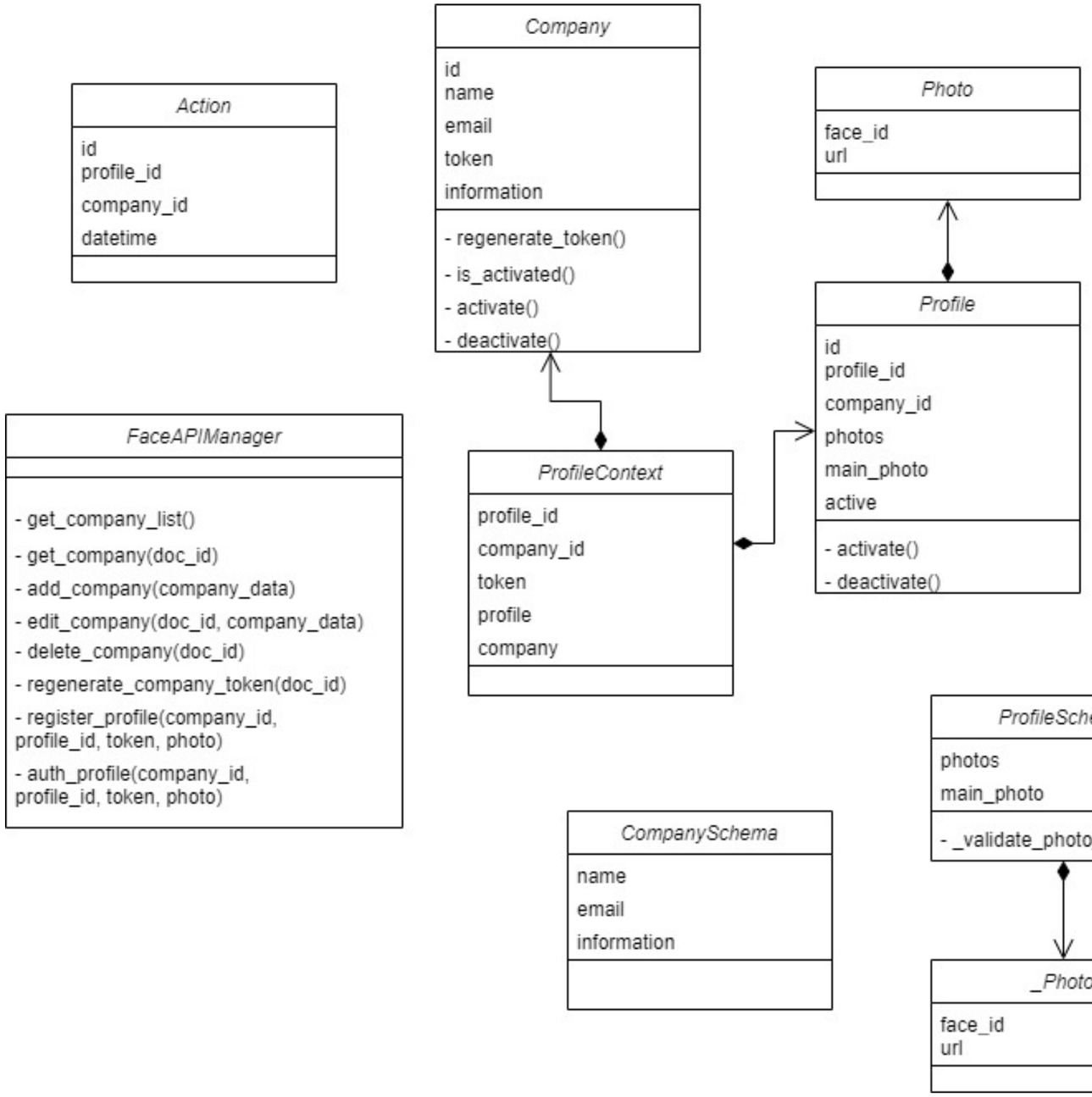
					ДП 6318.02.000 ССД	
Зм.	Арк.	№ докум.	Підп.	Дата		
Розроб.		Портяний І.С.			Схема структурна діяльності	Лит. / Маса / Масштаб
Перев.		Жданова О.Г.				Аркуш 1 / Аркушів 1
Т. Кон.					Підсистема розпізнавання та автентифікації обличь в системі оплати	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63
Н. Кон.		Проскура С.Л.				
Затв.		Жданова О.Г.				



					ДП 6318.03.000 ССВ		
					Схема структурна варіантів використань	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Портяний І.С.				Аркуш 1	Аркушів 1
Перевірив		Жданова О.Г.					
Т. кон.					*Підсистема розпізнавання та автентифікації обличь в системі оплати*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63*	
Н. кон.		Проскура С.Л.					
Затвердив		Жданова О.Г					

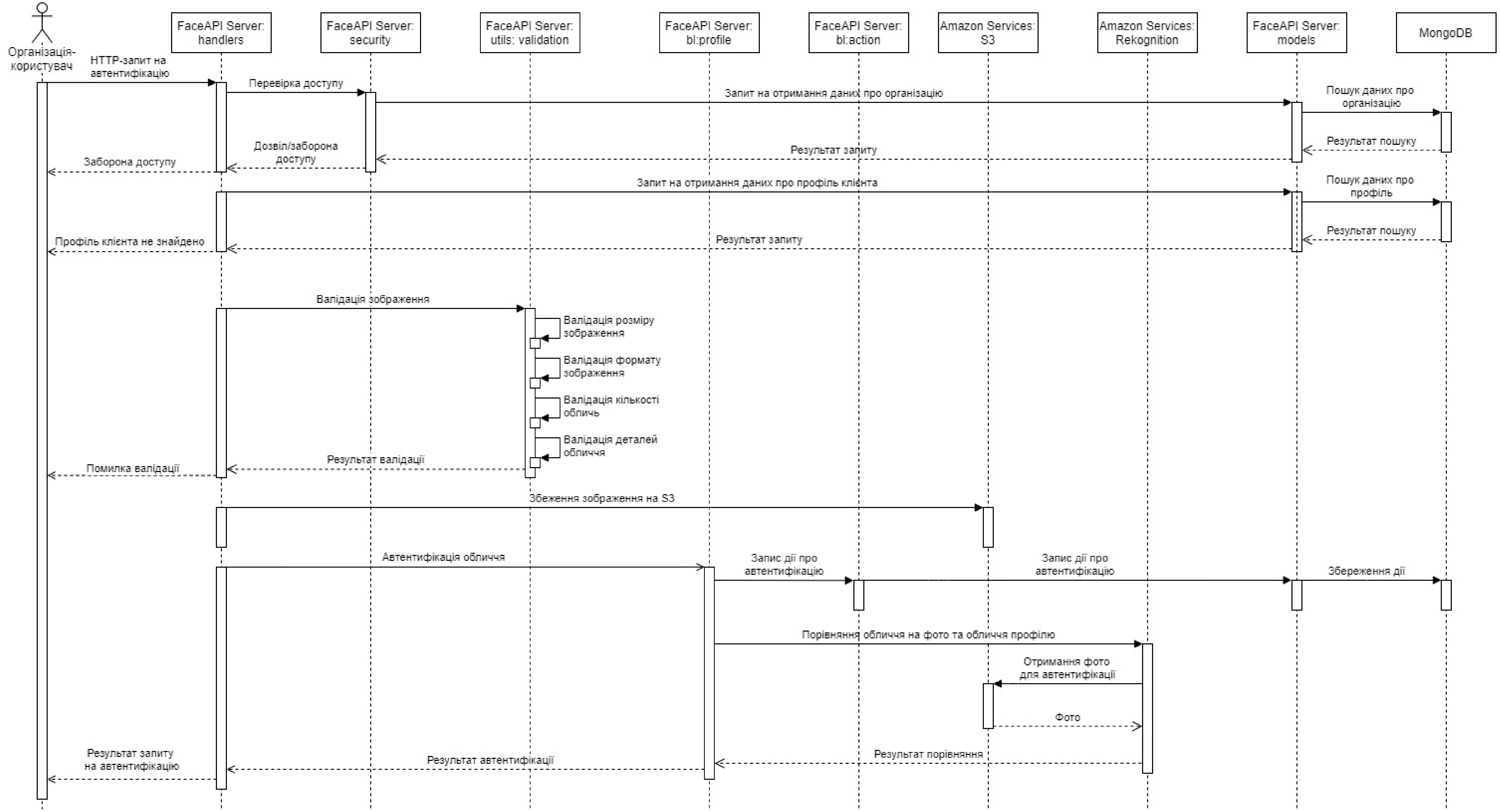
					ДП 6318.04.000 СБД	
						Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	*Схема бази даних*	
Розробив		Портяний І.С.				Аркуш 1 / Аркушів 1
Перевірив		Жданова О.Г.				
Т. кон.						
					Підсистема розпізнавання та автентифікації обличь в системі оплати	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63*
Н. кон.		Проскура С.Л.				
Затвердив		Жданова О.Г.				

ДП 6318.05.000 ССК



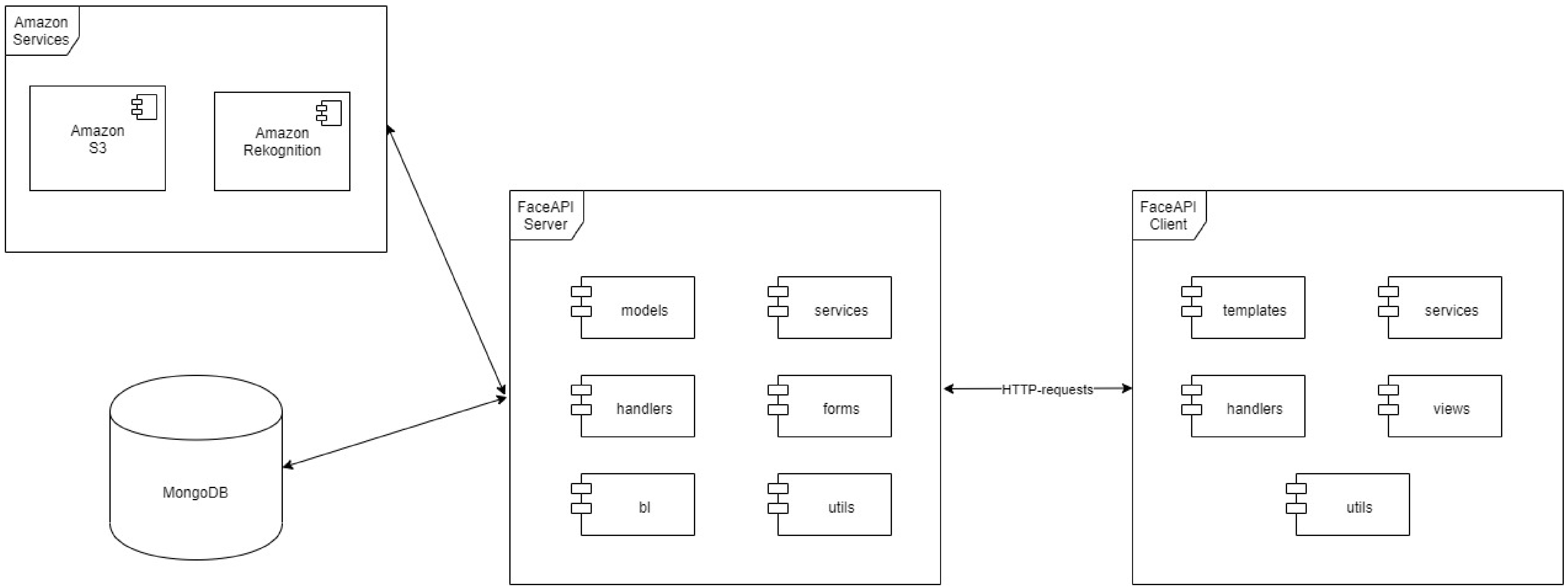
					ДП 6318.05.000 ССК		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення	Літера	Маса / Масштаб
Розробив		Портяний І.С.					
Перевірив		Жданова О.Г.				Аркуш 1	Аркушів 1
Т. кон.							
					Підсистема розпізнавання та автентифікації обличь в системі оплати	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63	
Н. кон.		Проскура С.Л.					
Затвердив		Жданова О.Г.					

ДП 6318.06.000 ССВ



Зм.	Арк.	№ документа	Підпис	Дата	ДП 6318.06.000 ССВ			
Розробив		Портяний І.С.			Схема структурна послідовності	Літера	Маса	Масштаб
Перевірив		Жданова О.Г.						
Т. кон.						Аркуш 1		Аркушів 1
Н. кон.		Проскура С.Л.			Підсистема розпізнавання та автентифікації обличь в системі оплати	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63		
Затвердив		Жданова О.Г.						

ДП 6318.07.000 ССК



					ДП 6318.07.000 ССК			
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна компонентів програмного забезпечення*	Літера	Маса	Масштаб
Розробив		Портяний І.С.						
Перевірив		Жданова О.Г.						
Т. кон.						Аркуш 1		Аркушів 1
					Підсистема розпізнавання та автентифікації облич в системі оплати	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-63*		
Н. кон.		Проскура С.Л.						
Затвердив		Жданова О.Г.						